10-2005

Investigation of a coupled duffing oscillator system in a varying potential field

Joseph Patrick O'Day

INVESTIGATION OF A COUPLED DUFFING OSCILLATOR SYSTEM IN A VARYING POTENTIAL FIELD

By

Joseph Patrick O'Day

A Thesis Submitted in
Partial Fulfillment of the
Requirements for the

MASTER OF SCIENCE

in

MECHANICAL ENGINEERING

Approved by:

Dr. Josef S. Török
Department of Mechanical Engineering

J. S. Török
(Thesis Advisor)

Dr. Agamemnon L. Crassidis
Department of Mechanical Engineering

Agamemnon Crassidis

Dr. Daniel B. Phillips
Department of Electrical Engineering

Daniel B. Phillips

Dr. Edward Hensel
Department Head of Mechanical Engineering

Edward Hensel

Department of Mechanical Engineering
Kate Gleason College of Engineering
Rochester Institute of Technology
October 2005

PERMISSION GRANTED

Title: **Investigation of a Coupled Duffing Oscillator System in a Varying Potential Field**

I, Joseph Patrick O'Day, hereby grant permission to the Wallace Library of the Rochester Institute of Technology to reproduce my thesis in whole or in part. Any reproduction will not be for commercial use or profit.

Date: 10/24/05 Signature of Author: Joseph P. O'Day

Thesis/Dissertation Author Permission Statement

Title of thesis or dissertation: INVESTIGATION OF A COUPLED DUFFING OSCILLATOR SYSTEM IN A VARYING POTENTIAL FIELD

Name of author: JOSEPH P. O'DAY
Degree: ~~EMEB~~ MS IN MECH. ENG.
Program: EMEB
College: KGCOE

I understand that I must submit a print copy of my thesis or dissertation to the RIT Archives, per current RIT guidelines for the completion of my degree. I hereby grant to the Rochester Institute of Technology and its agents the non-exclusive license to archive and make accessible my thesis or dissertation in whole or in part in all forms of media in perpetuity. I retain all other ownership rights to the copyright of the thesis or dissertation. I also retain the right to use in future works (such as articles or books) all or part of this thesis or dissertation.

Print Reproduction Permission Granted:

I, JOSEPH P O'DAY, hereby **grant permission** to the Rochester Institute Technology to reproduce my print thesis or dissertation in whole or in part. Any reproduction will not be for commercial use or profit

Signature of Author: Joseph P. O'Day Date: 10/24/05

Print Reproduction Permission Denied:

I, _____, hereby **deny permission** to the RIT Library of the Rochester Institute of Technology to reproduce my print thesis or dissertation in whole or in part.

Signature of Author: _____ Date: _____

Inclusion in the RIT Digital Media Library Electronic Thesis & Dissertation (ETD) Archive

I, JOSEPH P O'DAY, additionally grant to the Rochester Institute of Technology Digital Media Library (RIT DML) the non-exclusive license to archive and provide electronic access to my thesis or dissertation in whole or in part in all forms of media in perpetuity.

I understand that my work, in addition to its bibliographic record and abstract, will be available to the world-wide community of scholars and researchers through the RIT DML. I retain all other ownership rights to the copyright of the thesis or dissertation. I also retain the right to use in future works (such as articles or books) all or part of this thesis or dissertation. I am aware that the Rochester Institute of Technology does not require registration of copyright for ETDs.

I hereby certify that, if appropriate, I have obtained and attached written permission statements from the owners of each third party copyrighted matter to be included in my thesis or dissertation. I certify that the version I submitted is the same as that approved by my committee.

Signature of Author: Joseph P. O'Day Date: 10/24/05

Table of Contents

List of Figures

Abstract

Nonlinear systems are known to exhibit widely differing steady–state behaviors based on small modifications to the control parameters within the equations. These small modifications may be the difference between a chaotic output and a periodic output. Many investigators choose to study the varying behaviors through varying forcing conditions, specifically the forcing amplitude or frequency. However, from a linear vibration theory standpoint, systems are often "tuned" to minimize system response to a known forcing input by varying the strength of the damping and stiffness elements within the system. It may also be the case that the parameters governing the strengths of these elements are constant, but uncertain within a specific range. In these cases, it is more advantageous to understand how the response will vary based on these design parameters or uncertain constants.

The parameters defining the potential field for a nonlinearly coupled Duffing oscillator system were used as the control parameters in this study. The steady–state system response was investigated through the techniques of Poincaré maps, bifurcation diagrams, Lyapunov exponents and spectra, power spectra estimates, and phase portrait projections. The system of equations was integrated through a semi–discrete algorithm based on continuous transformation group theory, which improved the accuracy of the integrated trajectories and the accuracy of the Lyapunov exponents. Additionally the Poincaré maps, power spectra estimates, and phase portrait projections were animated to simplify the analysis of the varying parameters. The use of these animations saved countless hours of analysis time, and revealed details of the parameter–based variations that would not have been observed otherwise. This technique has not been used in any of the known literature.

Two separate forcing conditions were considered; synchronous sinusoidal forcing on each oscillator and nonsynchronous forcing. Each system exhibits a wide variety of nonlinear phenomena including period–doubling sequences, quasiperiodicity, and chaos. The existence of the merging of chaotic attractors and hysteresis was also confirmed. This study also suggests the hyperchaotic attractors and chaotic tori may be present under certain parameter combinations.

Acknowledgements

Foremost, I would like to thank Dr. (Dr.^{2}) Török for his guidance and encouragement in the classroom and throughout this thesis. His teaching style and personality serve to inspire those who are always a bit curious as to what comes next.

I would also like to thank the members of my committee who took time out of their schedules to review and comment on my work. And to many of the professors of the Mechanical Engineering department, I would like to express my gratitude for all that they have taught me, both in and out of the classroom.

I would like to acknowledge Bill Finch for helping me deal with any computer crisis that I was having at that current moment, (there were many!), and also, the staff of the Mechanical Engineering department for making my life easier over the course of my time at RIT.

I would like to thank my parents and family for always encouraging me to achieve as much as I can in whatever path I chose, and for always looking out for my best interests, even if I couldn't see them.

I fear where I'd be if Seth Kane and Brendan Shaughnessy didn't often remind me that sometimes I need to stop being "such an engineer." They were, and are, right.

Lastly I would like to thank Audrey Acuña for sharing in my thoughts, frustrations, and excitement over this thesis and everything else. Her support, understanding, and most of all, motivation was invaluable to my sanity and the completion of this work.

Chapter 1

Nonlinear Dynamics

The fundamental problem with differential equations of a nonlinear nature is the lack of methods for readily identifying explicit solutions. Typically employed methods for linear system solutions such as conversion to the Laplace domain, eigenvalue/eigenvector analysis, and the superposition principle no longer hold any merit with nonlinear differential equations. Unlike linear equations, the total solution will not be the sum of the solutions of the parts. This is not to say that those who wish to study nonlinear differential equations, and nonlinear dynamics in general, are at a loss when faced with an analysis, merely that they must look for solutions through other means. Employing semi-quantitative, geometric methods such as those developed by Poincaré, definitions and classifications of stability developed by Lyapunov, and some modified methods from linear analysis offer an alternative to identifying the type of dynamic behavior to be expected under a variety of conditions.

To first understand how to analyze the behavior of a system governed by a nonlinear differential equation, it is necessary to examine and understand the nature and some of the nuances of nonlinear systems in general.

1.1 State–Space Formulation

A system of equations is commonly placed into a form in which the most information can be garnered. One of the more common forms is the state–space formulation. For linear systems, an equation or system of equations is placed in this form by setting the

time derivative of the state space vector equal to the state matrix multiplied by the state vector.

$$\dot{\vec{x}} = \mathsf{A}\vec{x} \tag{1.1}$$

where $\vec{x}$ is an $n \times 1$ state vector and A is an $n \times n$ system matrix. Typically the number of these states is defined by the number of energy elements in the modeled system. The number of states also would be the degree of the differential equation. A simple harmonic oscillator (Figure 1.1) would have a state vector comprised of two states; x corresponding to the potential energy of the spring, and $\dot{x}$ corresponding to the kinetic energy of the mass. The corresponding differential equation would also be of second order as expected.

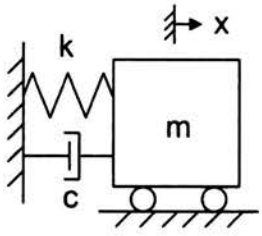


Figure 1.1: Simple harmonic oscillator

$$m\ddot{x} + c\dot{x} + kx = 0 \quad \Longleftrightarrow \quad \begin{pmatrix} \dot{x} \\ \ddot{x} \end{pmatrix} = \begin{bmatrix} 0 & 1 \\ -k/m & -b/m \end{bmatrix} \begin{pmatrix} x \\ \dot{x} \end{pmatrix} \tag{1.2}$$

1.2 Linear Systems

It is helpful to examine what can be obtained from the state–space formulation of linear systems, as some of the techniques can be directly applied or modified slightly for nonlinear systems.

1.2.1 Phase Space

It is rather evident that $\dot{\vec{x}}$ in Equation 1.1 can be thought of as a velocity vector. At every state $\vec{x}$, there is a corresponding magnitude and direction of the time rate of change of that state. If the state vector is thought of as a coordinate point in the x–$\dot{x}$ space, these velocities can be plotted for any range of coordinates. This space is known as the

phase space or phase plane if the system is two–dimensional, and the plot is known as a direction plot. The direction plot of the simple harmonic oscillator, neglecting the damping effects of the dashpot, is shown in Figure 1.2.

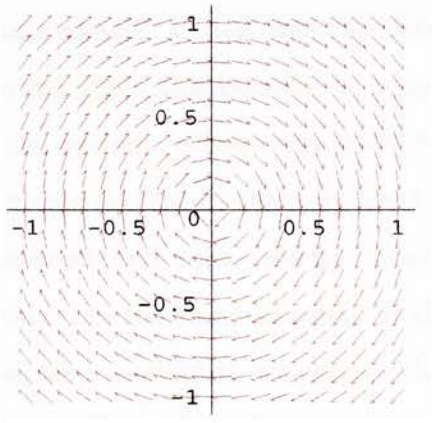


Figure 1.2: Direction plot of a simple harmonic oscillator (damping neglected)

Each coordinate can also be thought of an initial condition as well. Thus trajectories can be plotted for any initial condition. A collection of plotted trajectories in the phase space is known as a phase portrait. (Figure 1.3) The closed nature of the trajectories

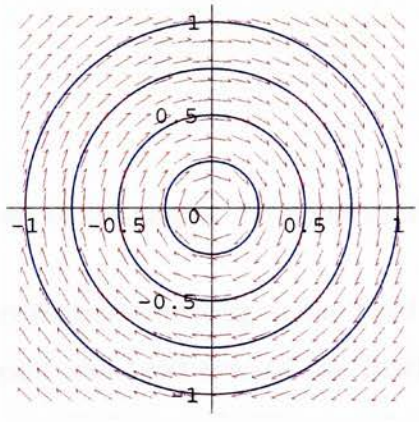


Figure 1.3: Phase portrait of the simple harmonic oscillator (damping neglected)

signifies that this system is repetitive or cyclic. In other words, the system behaves in a periodic manner.

1.2.2 Fixed Points

As shown in the previous section, the derivative of the state vector can be thought of a velocity having both a direction and a magnitude. But what if this velocity is zero for a certain coordinate in the state–space? If that was the initial condition of the system, no movement would be observed since the velocity is perpetually zero. These points are known as fixed points of the system, which correspond to steady–state or equilibria conditions. In the simple harmonic oscillator, the only fixed point has coordinates $(0, 0)$ corresponding to zero displacement and zero velocity. Since there is no forcing being applied to the mass, it will remain in the same location. If the positive damping term is added back into the simple harmonic oscillator, we can see that all trajectories will spiral toward the fixed point at $(0, 0)$ (see Figure 1.4). In this sense, the fixed point is considered

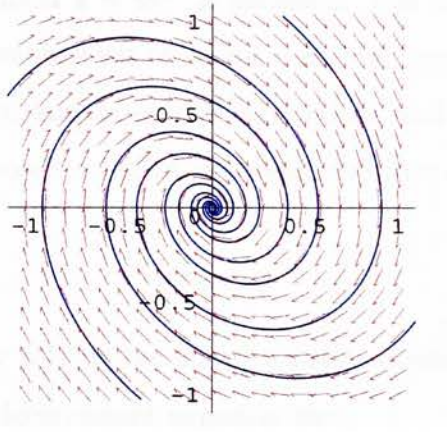


Figure 1.4: Phase portrait of the simple harmonic oscillator (damping included)

to be attracting, that is all trajectories starting at the fixed point will remain there, and slight deviations from the point will tend toward the fixed point and eventually reach it. This type of point is often called an *sink*. This point is also Lyapunov stable. If a fixed point is Lyapunov stable, all trajectories that start near the fixed point will remain there for all time, not just as $t \to \infty$. If any fixed point is attracting and Lyapunov stable, then it is considered to be asymptotically stable. Unstable fixed points can also exist. All trajectories starting at an unstable fixed point will remain there, but unlike a stable point, the slightest deviation from the point will cause the trajectory to expand outward. Conversely these point are also called *repellers* or *sources*. Finally, there can

be an "in–between" case, as we saw with the undamped harmonic oscillator (Figure 1.3). This fixed point is considered to be neutrally stable. It is Lyapunov stable, but it neither attracts nor repels nearby trajectories.

The simple harmonic oscillator belongs to a larger class of second–order autonomous systems.

$$\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \qquad \text{or} \qquad \begin{aligned} \dot{x} &= ax + by \\ \dot{y} &= cx + dy \end{aligned} \tag{1.3}$$

Recall that an autonomous system is one in which there is no explicit time dependence. The equations defining a non–autonomous system have time dependence. The simple harmonic oscillator would become non–autonomous if we added a sinusoidal forcing term. Many different types of fixed points can exist for this system depending on the choices of a, b, c, and d, or matrix A given in Equation 1.1. Using a common linear algebra approach, a solution of the form $\vec{x} = \vec{v}e^{\lambda t}$ is assumed. The form of this solution is such that it is a multiplicative combination of vector $\vec{v}$ and an exponential growth (or decay) rate λ. The values of λ are known as the eigenvalues and the vector $\vec{v}$ is known as the eigenvector. If this proposed solution is substituted into equation 1.3, the following equality results.

$$\mathsf{A}\vec{v} = \lambda\vec{v} \tag{1.4}$$

The values of λ are found by solving the characteristic equation $\det[\mathsf{A} - \lambda\mathsf{I}] = 0$ where I is the identity matrix. The determinant expands into

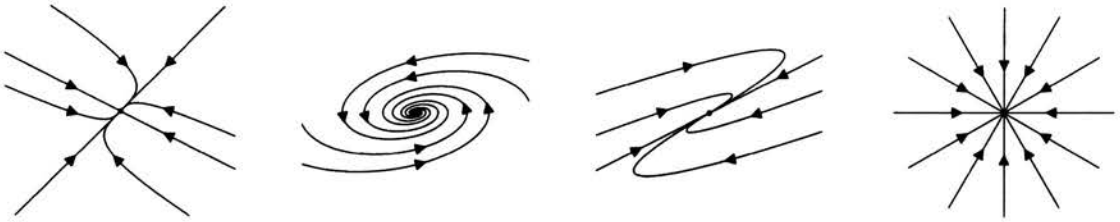
$$\lambda^2 - \tau\lambda + \Delta = 0 \tag{1.5}$$

where $\tau = trace(\mathsf{A}) = a + d$ and $\Delta = det(\mathsf{A}) = ad - bc$. The eigenvalues are then given by

$$\lambda_{1,2} = \frac{\tau \pm \sqrt{\tau^2 - 4\Delta}}{2} \tag{1.6}$$

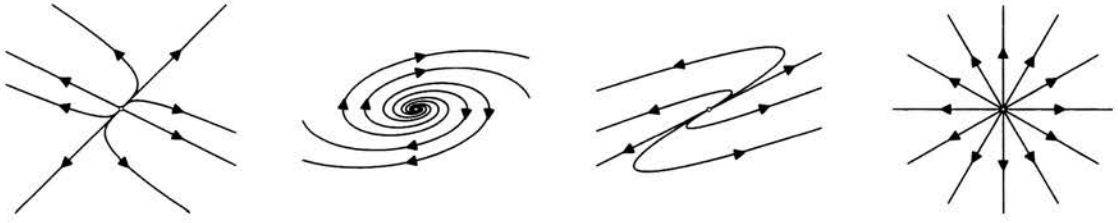
The nature of fixed points can be determined by examining the eigenvalues and thus the nature of the two parameters τ and Δ.

Figures 1.5, 1.6, and 1.7 show the various behaviors that are possible for a two–dimensional or second–order systems based on the eigenvalues. This scheme can be extended into higher dimensions or for higher order systems, but beyond three–dimensions,



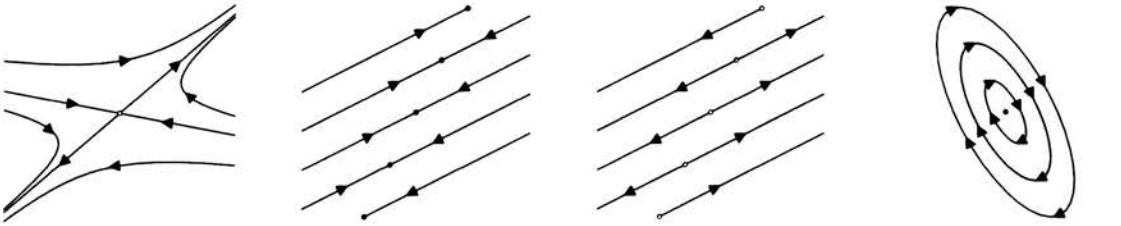
(a) (b) (c) (d)

Figure 1.5: Asymptotically stable fixed points corresponding to: (a) Stable node ($\lambda_1 < \lambda_2 < 0$); (b) Spiral ($\lambda_1 = \overline{\lambda_2}, \quad \mathrm{Re}\,\lambda_1 < \mathrm{Re}\,\lambda_2 < 0$); (c) Degenerate node ($\lambda_1 = \lambda_2 < 0$, one eigenvector); (d) Star node ($\lambda_1 = \lambda_2 < 0$, independent eigenvectors).



(a) (b) (c) (d)

Figure 1.6: Unstable fixed points corresponding to: (a) Unstable node ($0 < \lambda_1 < \lambda_2$); (b) Spiral ($\lambda_1 = \overline{\lambda_2}, \quad 0 < \mathrm{Re}\,\lambda_1 < \mathrm{Re}\,\lambda_2$); (c) Degenerate node ($0 < \lambda_1 = \lambda_2$, one eigenvector); (d) Star node ($0 < \lambda_1 = \lambda_2$, independent eigenvectors).



(a) (b) (c) (d)

Figure 1.7: Neutrally stable points corresponding to: (a) Saddle node ($\lambda_1 < 0 < \lambda_2$); (d) Center ($\lambda_1 = \overline{\lambda_2}, \quad \mathrm{Re}\,\lambda_1 = \mathrm{Re}\,\lambda_2 = 0$); (b) Non–isolated stable fixed points ($\lambda_1 = 0, \quad \lambda_2 < 0$); (c) Non–isolated unstable fixed points ($\lambda_1 = 0, \quad \lambda_2 > 0$).

the geometric representations become abstract. Figure 1.8 illustrates a classification scheme based on the parameters τ and Δ.

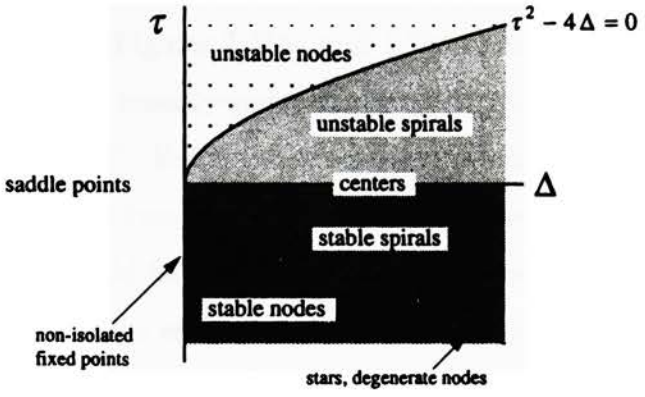


Figure 1.8: Stability types for second–order systems [1]

The fixed points occurring in Figures 1.5(a) and 1.6(a) are characterized by two distinct real eigenvalues and eigenvectors. One of these eigenvectors is associated with a faster rate of decay or expansion than the other and it is called the fast direction, while the other is called the slow direction. Trajectories approach a stable fixed point tangentially to the slow eigenvector. They also depart from an unstable fixed point tangentially.

The degenerate nodes portrayed in Figures 1.5(c) and 1.6(c) occur along the boundary between spirals and nodes. There is only one eigenvector associated with these types of fixed points. The fast and slow eigenvector can be considered to collide at this boundary into a single direction. Another consideration is that a trajectory approaching the fixed point could overshoot it similarly to a spiral, but when it reaches the other branch of the eigenvector is it forced to approach the fixed point tangentially.

The other types of node/spiral boundaries are the star nodes shown in Figures 1.5(d) and 1.6(d). Like the degenerate nodes, there is only one eigenvalue, but in this case the eigenvectors associated with it are linearly independent. Thus, they span the entire phase space and every trajectory is one that approaches or departs from a fixed point in a linear fashion.

The saddle node in 1.7(a) is an example of the neutrally stable point discussed above. It is evident from the figure that this fixed point is attracting in one direction and repelling

in the other. The attracting direction is called the unstable manifold while the repelling direction is called the stable manifold. For a two–dimensional system, these manifolds are lines. In higher dimensions, these manifolds can be planes or surfaces.

The non–isolated cases (Figures 1.7(b) and 1.7(c)) occur when the state equations defining the system are not linearly–independent, or equivalently that the matrix A is rank–deficient (Equation 1.3). For this type of system, one of the equations is a scalar multiple of the other. In the two–dimensional case, this results in a line of fixed points. In higher dimensions, it would be a plane or surface of fixed points.

Up until this point, with the exclusion of the non–isolated fixed points, all of the fixed points have had eigenvalues with a non–zero real part. The center, shown in Figure 1.7(d), has no real part of either eigenvalue. One consequence of this is the neutrality of the fixed point. Another is the relative fragility of this point compared to the others already discussed. In the previous examples, the stability of the point was persistent through small perturbations in the state–space. While a degenerate node could be changed into either a spiral or a node due to a small perturbation, the stability of the system would be unchanged. A center is different in that it can be changed into a stable or unstable spiral depending on the perturbation. All of the other types of phase portraits of fixed points are structurally stable, that is their stability is unchanged through small perturbations. The fixed points associated with structurally stable phase portraits are called hyperbolic points. The center does not fall into this category.

1.3 Nonlinear Systems

This last point about the structural stability begins the transition into the nonlinear realm. If the perturbation is of a nonlinear nature, the stability of the local phase portrait can still be assured as long as the perturbation is small. This is known as the *Hartman-Grobman Theorem* [1]. Thus, nodes, spirals, and saddle points can all occur in nonlinear systems and they can be classified by the same stability requirements. However, a nonlinear system will not fall into the same formulation as that in Equation 1.3 because there are no provisions for dealing with trigonometric terms, squared terms, cubic terms,

or anything else of a nonlinear nature. If the matrix A is viewed in a different light, there is indeed a method for dealing with nonlinearities. Equation 1.3 can be reformulated into

$$\dot{\vec{x}} = \vec{F}(\vec{x}) \tag{1.7}$$

with $F_1(x_1, x_2) = ax_1 + bx_2$ and $F_2(x_1, x_2) = cx_1 + dx_2$. Then matrix A can be written as

$$\mathsf{A} = \begin{bmatrix} \frac{\partial F_1}{\partial x_1} & \frac{\partial F_1}{\partial x_2} \\ \frac{\partial F_2}{\partial x_1} & \frac{\partial F_2}{\partial x_2} \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \tag{1.8}$$

The matrix A is simply the Jacobian matrix or the rate of change of each state due to each state. The Jacobian matrix can be easily defined for linear and nonlinear systems alike. The difference is that the Jacobian of a nonlinear system will depend on the location in which is it calculated, whereas the Jacobian of a linear system is constant through the entire phase space. Thus the stability of each fixed point will need to be calculated at that specific fixed point. It will be shown later that the Jacobian can also help to define another type of nonlinear stability.

One of the goals of a nonlinear autonomous system analysis is characterizing the behavior of each fixed point in the phase space. If multiple attracting fixed points exist, they will compete for the attraction of each and every initial condition in the space. But it is not possible for every initial condition to end up at every attracting fixed point. Thus there must be a certain range of initial conditions that are attracted to each fixed point. These ranges are known as basins of attraction. In Figure 1.9, the basins of attraction for each fixed point are separated by the stable manifold of the saddle point at $(1, 1)$.

1.3.1 Bifurcations

The question still remains as to what could cause a change in the stability of a fixed point, or a group of fixed points in a nonlinear system. Bifurcation theory can be used to explain this stability change. A bifurcation is defined as a qualitative change in the stability of the system. For the simple harmonic oscillator, bifurcations can occur if the damping or the stiffness is changed from positive to negative. Thus, the damping and

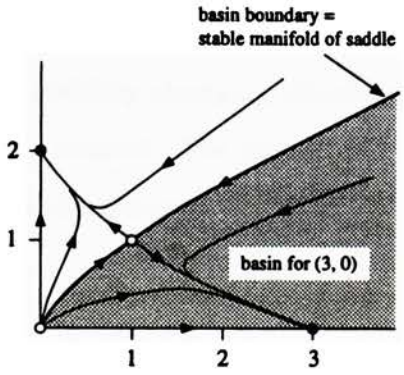


Figure 1.9: Basins of attraction for a second–order system [1]

stiffness values are control parameters of the system. The values at which the control parameters change the qualitative behavior of the system are called bifurcation points. A diagram plotting the change in stability against the control parameter is known as a bifurcation diagram. While there are two bifurcations of the simple harmonic oscillator, they are entirely different in nature.

Saddle–Node Bifurcation

A saddle–node bifurcation occurs when a neutrally stable fixed point appears and then divides into a stable and unstable fixed point. The following formula is the prototypical example of a saddle–node bifurcation in two–dimensions.

$$\begin{aligned} \dot{x} &= r + x^2 \\ \dot{y} &= -y \end{aligned} \tag{1.9}$$

In this case $F_1 = r + x^2$ is simply a vertically offset parabola in the x–direction. A fixed point is created when this parabola intersects with the x–axis. The y–direction is simply exponential damping and the y–coordinate of the fixed point, if it exists, is zero. The key is the behavior of the x–direction parabola. For large offsets r, there is no intersection, but as the offset becomes smaller, the parabola will eventually become tangent to the x–axis. This tangent point is the neutrally stable point. As the offset is decreased further, the apex of the parabola will dip below the axis and the parabola will touch the axis in two locations—the stable and unstable fixed points.

Transcritical Bifurcation

A transcritical bifurcation is a stability changing bifurcation, unlike the saddle–node bifurcation where fixed points are created. The prototypical example in this case is again a parabola in the x–direction and exponential damping in the y–direction.

$$\begin{aligned} \dot{x} &= rx - x^2 \\ \dot{y} &= -y \end{aligned} \tag{1.10}$$

The the y–coordinate of the fixed point is again always zero. In the x–direction, $F_1 = rx - x^2$ is an inverted parabola that is constrained to pass through $x = 0$. For values of r less than zero, the x–coordinate of the stable point is $x = 0$ and the unstable coordinate is $x = r$. As r is increased the parabola "swings" around $x = 0$ and the coordinate of the unstable fixed point moves toward the coordinate of the stable fixed point until they collide when $r = 0$ leaving a neutrally stable point at $x = 0$. As r is increased further, the fixed point at $x = 0$ is rendered unstable and a stable fixed point now exists at $x = r$.

Pitchfork Bifurcations

Pitchfork bifurcations are a combination of saddle–node and transcritical bifurcations in the sense that fixed points are both created and existing fixed points change stability. The prototypical example here is a cubic polynomial in the x–direction and exponential damping in the y–direction.

$$\begin{aligned} \dot{x} &= rx - x^3 \\ \dot{y} &= -y \end{aligned} \tag{1.11}$$

The behavior of the cubic polynomial is what drives the bifurcation. For $r < 0$ the cubic $F_1(x) = rx - x^3$ passes through $x = 0$ in one location with a negative slope indicating that the corresponding fixed point is stable. As r is increased to $r = 0$, F_1 now meets $x = 0$ tangentially. As r become positive, the slope of F_1 at $x = 0$ becomes positive indicating a change in the stability of the fixed point. Also now there are two additional stable fixed points at $x = \pm\sqrt{r}$. Thus an originally stable fixed point becomes unstable at the same time two new stable fixed points are created. This is known as a supercritical

pitchfork bifurcation. The other type is when the cubic term is positive.

$$\begin{aligned} \dot{x} &= rx + x^3 \\ \dot{y} &= -y \end{aligned} \tag{1.12}$$

Three fixed points originally exist when $r < 0$; a stable point surrounded by two symmetric unstable points. As r is increased to zero, all three points collide. The two symmetric unstable points are annihilated and the original stable fixed point is rendered unstable. This is known as a subcritical pitchfork bifurcation. Generally, pitchfork bifurcations occur in models with symmetry. In the simple harmonic oscillator, a supercritical bifurcation occurs as the spring stiffness is changed from positive to negative, or as the spring is changed from being initially stretched to initially compressed.

1.3.2 Limit Cycles

Nonlinear systems have the ability to be self–excited. That is, unlike damped linear systems which require a sinusoidal forcing term to be cyclic in nature, damped nonlinear systems can result in cyclic behavior merely from initial conditions. Perhaps the most famous example of this is Van der Pol's oscillator

$$\ddot{x} + \mu(x^2 - 1)\dot{x} + x = 0 \tag{1.13}$$

where $\mu \geq 0$ is a parameter. The nonlinear damping term $\mu(x^2 - 1)\dot{x}$ is negative for $|x| < 1$ and positive for $|x| > 1$. The damping term will act to decay oscillations with an amplitude larger than one (stable spiral), but small amplitude oscillations will be increased (unstable spiral). Intuitively, there must be a certain amplitude that is small enough not to decay any further, but large enough not to be increased. The trajectory associated with this amplitude is called a limit cycle. A limit cycle is defined as an isolated closed trajectory [1]. That means that the center discussed in Section 1.2.2 does not have a limit cycle because there are an infinite number of closed orbits surrounding the center. Limit cycles have the same type of stability classification as points. A limit cycle can be attracting and is referred to as stable. It can also be repelling and called unstable. The neutrally stable notion of points is replaced by half–stable for limit cycles. A half–stable limit cycle is attracting in one radial direction and repelling in the other.

An example of the stable limit cycle in Van der Pol's equation (Equation 1.13) is shown in Figure 1.10. Notice how trajectories beginning outside and inside of the limit cycles are attracted toward it.

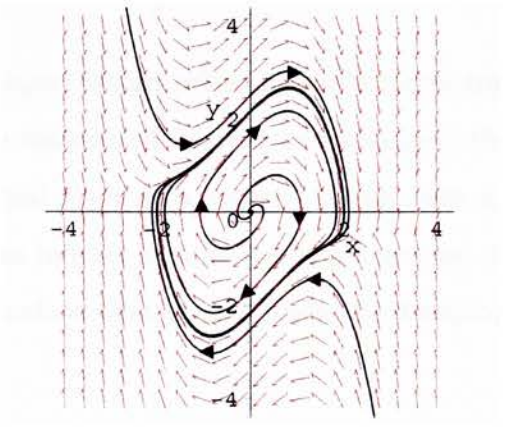


Figure 1.10: Stable limit cycle in Van der Pol's equation, $\mu = 1$

1.3.3 Bifurcations, Part II

There are also bifurcations that can occur between limit cycles, or limit cycles and fixed points. Similarly to the way that fixed points can collide and either annihilate or change stability, the interaction between multiple limit cycles or limit cycles and fixed points can produce stability altering results.

Hopf Bifurcation

A Hopf bifurcation occurs when a pair of complex conjugate eigenvalues crosses into the right–half plane. There are three types of Hopf bifurcations; supercritical, subcritical, and degenerate. A supercritical Hopf bifurcation occurs when a stable spiral changed to an unstable spiral surrounded by a small amplitude limit cycle. The subcritical bifurcation occurs when an unstable limit cycle shrinks down and absorbs a stable spiral thus changing it to an unstable spiral. This type of bifurcation is associated with hysteresis effects.

Hysteresis is an irreversibility along the parameter space due to the existence of multiple solutions for a certain range of the space. Varying the parameter positively ensures that the solution will tend to a certain solution branch and then jump to the

negative branch at a critical value. Varying the parameter negatively ensures that the solution will stay on the negative branch and then jump to the positive branch at a different critical value. Because of this jumping behavior, hysteresis is also called *jump phenomena.*

The degenerate bifurcation occurs when a stable spiral transitions through a center to an unstable spiral. It is considered degenerate because in the intermediary case, there is no limit cycle but instead there is a center. Recall that a limit cycle is isolated by definition. A center has an infinite number of cycles around it. In the simple harmonic oscillator, this is the bifurcation that occurs when the damping is changed from positive to negative.

Saddle–Node Bifurcation of Cycles

As shown in the first section on bifurcations, a saddle-node bifurcation occurs when a neutrally stable point is created and then divides into a stable and unstable point. This can also occur between cycles. A neutrally stable limit cycle can be created and then divide into a stable and unstable limit cycle.

Homoclinic Bifurcation

A homoclinic bifurcation occurs when a limit cycle becomes closer and closer to a saddle point and then finally touches. As the limit cycle touches the saddle, a homoclinic orbit results where the unstable manifold loops around becoming the stable manifold. Any further increase in the parameter causes the unstable manifold not to connect to the saddle but to diverge outward.

Period–Doubling Bifurcation

Thus far, the majority of bifurcations have occurred in autonomous systems. Period–doubling bifurcations can occur in both non–autonomous systems and discrete nonlinear systems such as the logistic map [1]. In the non–autonomous case, a period–doubling bifurcation occurs when a stable period–n orbit loses stability and a stable period–$2n$ orbit appears. This can be thought of as a supercritical pitchfork bifurcation, but with

the points representing orbits instead of fixed points. Successive period–doubling is a common route to chaotic behavior for a variety of systems.

1.3.4 Torus–Space and Quasiperiodicity

Another type of phenomena that does not exist in linear systems is quasiperiodicity. This type of behavior occurs in coupled systems with two distinct frequency components ω_1 and ω_2. If the relation ω_1/ω_2 is rational, that is $\omega_1/\omega_2 = p/q$ for some integers p, q, then the trajectories will be closed orbits. This is because the portion of the solution with frequency ω_1 will complete p orbits about the θ_1 space in the same amount of time that the portion with frequency ω_2 will complete q orbits about the θ_2 space. The behavior can be likened to the flower–shaped trajectories that could be created with the original Spirograph® toy. The space defined by θ_1 and θ_2 is known as torus–space, in this case a 2–torus. This behavior and coordinate scheme can be generalized to n–frequencies on an n–torus.

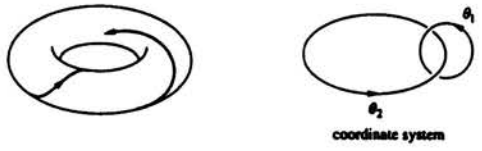


Figure 1.11: Torus–space and coordinate system [1]

The relation ω_1/ω_2 is not necessarily rational. If it was irrational, then the trajectories would not form closed orbits. They would be nearly periodic but not quite. The returning trajectory one cycle later would be separated from the original trajectory by an arbitrarily small amount. This behavior is known as quasiperiodicity. If the Spirograph developed a significant amount of play in the gears due to wear, then the shapes created would be like quasiperiodic trajectories. Quasiperiodicity is often considered to be a precursor to chaos.

1.3.5 Chaos

Chaos is described as aperiodic long–term behavior that has a sensitive dependence on initial conditions [1]. Chaos is not random, however. Chaotic behavior results from

equations that are clearly defined or deterministic, and chaos itself has an underlying structure. The significant dependence on initial conditions makes long–term behavior completely unpredictable. In Figure 1.12, the shaded box in the left figure represents the uncertainty in an initial condition. At some time later, this uncertainty has grown due to the chaotic nature of the system.

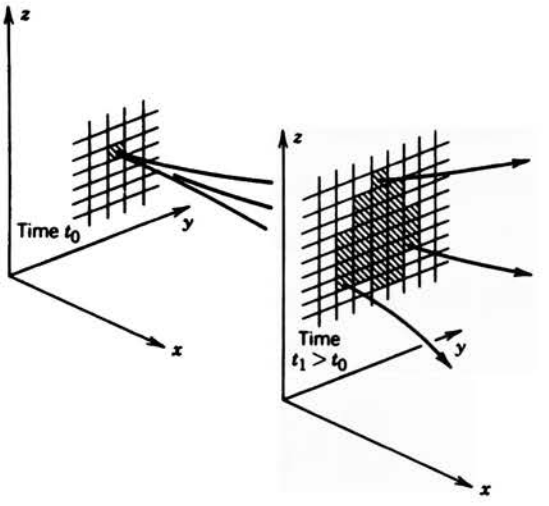


Figure 1.12: Loss of precision in a chaotic system [2]

Lyapunov exponents are one measure of the unpredictability or the degree of chaos in a system. Lyapunov exponents quantify the expansion or contraction of nearby trajectories in the phase space. A positive exponent would correspond to expansion and the sensitive initial condition dependence associated with chaos. A negative exponent implies some evolution toward a common value, likely an attracting solution. Consider a point $\vec{x}(t)$ and a nearby point $\vec{x}(t) + \vec{\delta}(t)$ on a chaotic attractor at some time t, where $\vec{\delta}$ is a small separation vector (Figure 1.13).

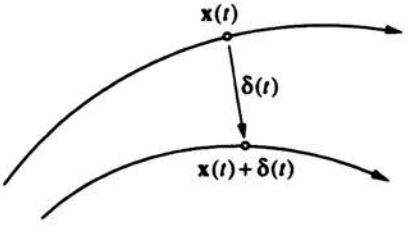


Figure 1.13: Nearby trajectories in the phase space [1]

Mathematically, the type of behavior to be expected is

$$\vec{\delta}(t) \sim \vec{\delta}(0)e^{\vec{\lambda}t} \tag{1.14}$$

where $\vec{\lambda}$ is a vector comprised of the Lyapunov exponents. The number of Lyapunov exponents corresponds to the number of system states. Geometrically, $\vec{\delta}(t)$ can be thought of as a vector containing the magnitudes of the directions that comprise a hypervolume of initial condition perturbations. This hypervolume will shrink or expand along each of these directions as the system evolves, and each Lyapunov exponent quantifies the change in its respective direction. The behavior of the hypervolume will be dominated by the largest value of the Lyapunov exponent.

The question still remains at to how long in time will prediction of the system be valid? Equation 1.14 can be rearranged as follows

$$t_{horizon} = \frac{1}{\lambda} \ln \frac{\|\delta\|}{\|\delta_0\|} \tag{1.15}$$

where $t_{horizon}$ is when the prediction begins to fail. $\|\delta\|$ can be considered to be a tolerance on the initial "nearness" of the trajectories. The following example illustrates just how sensitive the initial conditions can be [1]. Say the tolerance is $\|\delta\| = 10^{-3}$. The accuracy of the initial condition is specified to $\|\delta_0\| = 10^{-7}$. An improved accuracy of the initial condition is $\|\delta_0\| = 10^{-13}$, or a million times more accurate. What is the relative increase in prediction time?

$$t_{horizon} \simeq \frac{1}{\lambda} \ln \frac{10^{-3}}{10^{-7}} = \frac{4 \ln 10}{\lambda} \tag{1.16}$$

The improved prediction

$$t_{horizon} \simeq \frac{1}{\lambda} \ln \frac{10^{-3}}{10^{-13}} = \frac{10 \ln 10}{\lambda} \tag{1.17}$$

Thus the increased prediction time is only $10/4 = 2.5$ times longer for an increase in the accuracy of a million times. This example clearly illustrates the importance of accurate initial conditions when dealing with a chaotic system.

Chapter 2

Nonlinear Analysis Methods

This chapter introduces many of the alternative methods that are used to analysis nonlinear system behavior. Section 2.1 introduces a numerical integration scheme that has not been typically employed to solve nonlinear differential equations. Section 2.2 presents the method of Poincaré maps. The phase–space representation of the system dynamics is simplified through the use of this technique. An alternative method for bifurcation diagrams based on non–autonomous systems is given in Section 2.3. This method makes use of the Poincaré mapping technique. The concept of Lyapunov exponents is further refined and calculation methods are presented in Section 2.4. The method for calculating the exponents is also modified from the normal procedure to reflect the alternative integration scheme. It is hypothesized that this alternative method yields more accurate results than the typical linearized approach. This section also details how the dynamics of the system can be characterized through the Lyapunov exponents. Finally, estimates of the frequency content of integrated trajectories are discussed in Section 2.5.

2.1 Numerical Integration

It was previously stated in Chapter 1 that finding explicit solutions to nonlinear differential equations is rarely possible. In most cases, the investigator of a nonlinear system is forced to solve the equations using a numerical integration scheme. The two major considerations when choosing an integration scheme are the computational speed and numerical accuracy of the solver. The fourth–order Runge–Kutta method is typically

employed. A much more elegant and potentially more accurate approach is one stemming from continuous transformation group theory [3, 4, 5]. This method focuses on developing power series solutions to nonlinear differential equations (see [4] for a full derivation).

A transformation in $\mathbb{R}^n$, where n is the dimension of the state–space, is given by

$$x_i^a = \phi_i\left(x_1, x_2, \ldots, x_n; a\right) \quad i = 1, 2, \ldots, n \tag{2.1}$$

where a is an element in a continuous transformation group. Multiple transformations are considered to be a composition of functions. These transformations could be rotations, dilations, translations, etc. For solving differential equations, the parameter a is chosen to be the time variable, t. Thus our collection of state equations can be thought of as a combination of stretching, rotating, and translating the state variables through time.

2.1.1 Differential Operator and Series Solution

A differential operator U is defined such that

$$U = \frac{\partial}{\partial x_1}\frac{\mathrm{d}x_1}{\mathrm{d}t} + \frac{\partial}{\partial x_2}\frac{\mathrm{d}x_2}{\mathrm{d}t} + \ldots + \frac{\partial}{\partial x_n}\frac{\mathrm{d}x_n}{\mathrm{d}t} \tag{2.2}$$

where x_i are the state variables. The derivatives $\mathrm{d}x_1/\mathrm{d}t, \mathrm{d}x_2/\mathrm{d}t, \ldots, \mathrm{d}x_n/\mathrm{d}t$ simply represent the time derivatives of the state variables or the state equations themselves.

$$\begin{aligned} \frac{\mathrm{d}x_1}{\mathrm{d}t} &= F_1 \\ \frac{\mathrm{d}x_2}{\mathrm{d}t} &= F_2 \\ &\vdots \\ \frac{\mathrm{d}x_n}{\mathrm{d}t} &= F_n \end{aligned} \tag{2.3}$$

U is known as an infinitesimal transformation operator or infinitesimal generator. From the standpoint of analytical mechanics, an infinitesimal transformation represents a virtual displacement [4].

Consider a system of the form

$$\left(\dot{x_1}, \dot{x_2}, \ldots, \dot{x_n}\right) = F_i\left(x_1, x_2, \ldots, x_n\right) \quad i = 1, 2, \ldots, n \tag{2.4}$$

F_i is a function of parameter t by definition, and it can be expanded in a power series about $t = 0$, or the initial time. Using the definition of U results in

$$F_i\left(x_1^t, x_2^t, \ldots, x_n^t\right) = F_i\left(x_1, x_2, \ldots, x_n\right) + \frac{t}{1!}UF_i + \frac{t^2}{2!}U^2F_i + \cdots = \sum_{k=0}^{\infty}\frac{t^k}{k!}U^kF_i \tag{2.5}$$

or for each state variable

$$x_i^t = x_i + \frac{t}{1!}Ux_i + \frac{t^2}{2!}U^2x_i + \cdots = \sum_{k=0}^{\infty}\frac{t^k}{k!}U^kx_i \tag{2.6}$$

This solution can be formulated as a Lie series, which is an exponential of a differential operator as shown above.

A series solution is not very practical unless the series is convergent, so the series is typically truncated to a certain number of terms. The global truncation error in this approximation is given as $\sim O(h^n)$ where h is the time step and n is the order of approximation. The fourth–order Runge–Kutta algorithm has an error of $\sim O(h^4)$. A Lie series approximation of order greater than four should exceed the accuracy of the Runge–Kutta method.

2.1.2 Infinitesimal Generator Computation

The powers of the U operator are computed by composition.

$$U^{n+1}x_i = U\left[U^nx_i\right] \tag{2.7}$$

The following example illustrates how this procedure is used [6]. Consider the nonlinear differential equation

$$\dot{x} = 1 - x^2 \quad \text{with} \quad x(0) = x_0 \tag{2.8}$$

Here $F = 1 - x^2$. Thus the infinitesimal generator is given by

$$U = \left(1 - x^2\right)\frac{\partial}{\partial x} \tag{2.9}$$

Successive application of U results in

$$U^1x = F\frac{\partial}{\partial x}(x) = 1 - x^2 \tag{2.10}$$

$$U^2x = F\frac{\partial}{\partial x}\left(1 - x^2\right) = \left(1 - x^2\right)(-2x) \tag{2.11}$$

$$U^3x = U\left(-2x + 2x^3\right) = F\frac{\partial}{\partial x}\left(-2x + 2x^3\right) = \left(1 - x^2\right)\left(-2 + 6x^2\right) \tag{2.12}$$

The series solution with a time step of h and initial condition x_0 would be given by

$$x(h) = x_0 + h\left(1 - x_0^2\right) + \frac{h^2}{2!}\left(-2x_0 + 2x_0^3\right) + \frac{h^3}{3!}\left(-2 + 8x_0^2 - 6x_0^4\right) + \cdots \tag{2.13}$$

A third–order approximation would be given by

$$x(h) = x_0 + h\left(1 - x_0^2\right) + \frac{h^2}{2!}\left(-2x_0 + 2x_0^3\right) + \frac{h^3}{3!}\left(-2 + 8x_0^2 - 6x_0^4\right) \tag{2.14}$$

It can be seen that the calculation of the infinitesimal generators can be rather tedious even for a first–order system. The ease and accuracy in which they are produced can be greatly enhanced through the use of a symbolic mathematics program [3, 6, 7].

2.1.3 Non–autonomous Considerations

As one last consideration, note that the above example did not have any explicit time dependence, i. e. it is autonomous. A slight modification is required if the equations are non–autonomous since the infinitesimal generator cannot explicitly depend on the time variable [4]. This can be avoided because any non–autonomous system in $\mathbb{R}^n$ can be converted into an equivalent system in $\mathbb{R}^{n+1}$,

$$\dot{\vec{x}} = f(x,t), \quad \vec{x}(0) = \vec{x}_0 \qquad \Longrightarrow \qquad \begin{array}{ll} \dot{\vec{x}} = f(x,\tau), & \vec{x}(0) = \vec{x}_0 \\ \dot{\tau} = 1, & \tau(0) = 0 \end{array} \tag{2.15}$$

The series expansion of τ is then given by

$$\tau^t = \tau + tU\tau + \frac{t^2}{2!}U^2\tau + \frac{t^3}{3!}U^3\tau + \ldots \tag{2.16}$$

In this case $U = \partial/\partial t$. By inspection, $U\tau = 1$ and $U^n\tau = 0$ for $n > 1$ which results in

$$\tau(t) = t \tag{2.17}$$

which is simply a identity transformation on the variable t.

2.2 Poincaré Maps

The last point about the conversion of nth-order non–autonomous systems into an $(n+1)$th-order autonomous system brings up an interesting predicament in terms of

fixed points. Since the last state equation will always be a constant, the last state (time) can never be zero, and thus no fixed points exist for the system.

Fixed points can exist for non–autonomous systems, but not in the traditional sense. If the time dependence given in the left equation in Equation 2.15 is of a periodic nature with period T, then it can be re–written as

$$\begin{aligned} \dot{\vec{x}} &= f(\vec{x}, \theta T/2\pi), \quad & \vec{x}(t_0) &= \vec{x}_0 \\ \dot{\theta} &= 2\pi/T, & \theta(t_0) &= 2\pi t_0/T \end{aligned} \tag{2.18}$$

Since f was originally periodic with period T, the new system (Equation 2.18) is periodic with period 2π. Instead of the state equations being defined in $\mathbb{R}^{n+1}$ Euclidean space, they are now defined in the cylindrical space $\mathbb{R}^n \times S^1$ where $S^1 = [0, 2\pi)$ is a circle. An n-dimensional hyperplane $\Sigma \in \mathbb{R}^n \times S^1$ is defined as

$$\Sigma = \{(\vec{x}, \theta) \in \mathbb{R}^n \times S^1 : \theta = \theta_0\} \tag{2.19}$$

Every T seconds, the trajectory will intersect this hyperplane. The resulting map P_N is defined as

$$P_N(x) = \phi_{t_0+T}(x, t_0) \tag{2.20}$$

P_N is known as a Poincaré map. P_N can be considered a forward–advance or T-advance mapping [8, 9]. It can also be likened to a sampling the trajectories every T seconds. The same type of mapping can be done on autonomous systems, but the method is more complicated in that the choice of Σ is not as obvious and can lead to various complications [9].

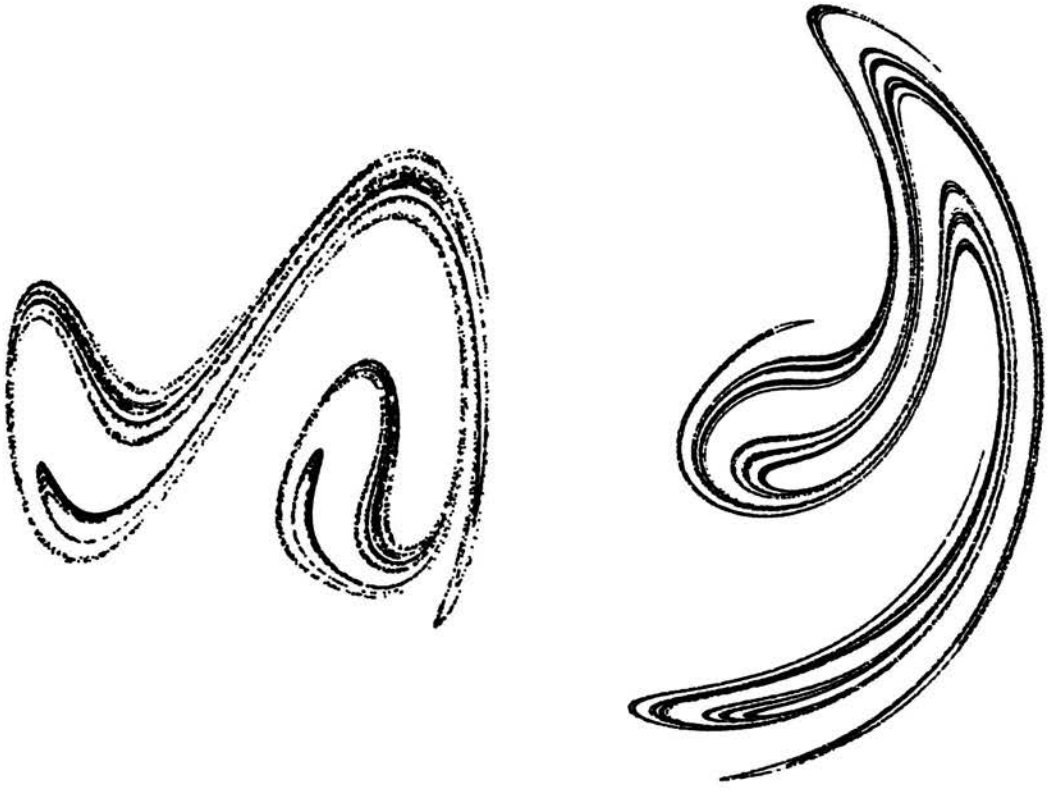
Fixed points can exist on the Poincaré map but they correspond to periodic solutions, not equilibria points. A periodic solution with period T will have exactly one unique point on the map.

$$\begin{aligned} \vec{x}(t+T) &= \vec{x}(t) \\ \phi_T(\vec{x}(t+T)) &= \phi_T(\vec{x}(t)) = \vec{x}^* \end{aligned} \tag{2.21}$$

A period-n solution will have exactly n unique points. The period–doubling bifurcations introduced in Section 1.3.3 should be easy to detect on the Poincaré map because a single point should split into two separate points at the bifurcation. Two points should split into four and so on and so forth.

Quasiperiodicity will be represented by a closed curve of points. This occurs because every for every period, the trajectory returns to a point arbitrarily close to the preceding point. If this behavior is captured for enough cycles, eventually the points will produce a closed curve.

The Poincaré map of chaos is the prototypical example of the mapping technique. These maps have beautifully intricate s–shaped curves (Figures 2.1(a) and (b)). Some of the structure of chaos can be seen through the use of this technique. Note the self–similar fractal–like structure in the loops.



(a) Duffing's equation [2] **(b)** Japanese attractor [10]

Figure 2.1: Chaotic Poincaré maps

2.3 Bifurcation Diagrams

Since there are no fixed points in the traditional sense, a bifurcation diagram consisting of the changes in the stability of said fixed points would not make much sense. Another

type of bifurcation diagram can be created which makes use of the Poincaré maps. In this case, the qualitative behavior of the system can be shown against a varying control parameter. Obviously, the entire n–dimensional map cannot be plotted against the varying control parameter, so typically one state variable is chosen. The procedure for creating a bifurcation diagram is described next. The system is integrated from an initial condition through any transient behavior for a certain value of the control parameter. The system is then integrated for a sufficient enough number of cycles to define the Poincaré map. The resulting behavior captured by the Poincaré map, whether it be periodic, quasiperiodic, or chaotic, is then used to create a vertical "slice" of the bifurcation diagram. This procedure is shown pictorially in Figure 2.2.

To avoid discontinuities in the bifurcation diagram, the final system state of the previously iterated value of the control parameter is chosen as the initial condition for the next system integration with the new value of the control parameter. This is to ensure that the resulting Poincaré map is somewhat similar to the last. More formally, this is done to ensure the new initial conditions are in the same basin of attraction as the old initial conditions, and that the same attracting solution is being reached.

2.4 Lyapunov Exponents

Lyapunov exponents were first introduced in Section 1.3.5, however the equations shown were merely definitions and have little relevance toward practical use. The method outlined in this section follows that of [9, 11]. As mentioned previously, Lyapunov exponents can be thought of as the expansion or contraction of vectors defining a hypervolume in the state–space. Let this hypervolume be denoted by B. This hypervolume can also be considered as a vector space spanned by the state variables. The rows of B are then the basis vectors of the space. A *fiducial* trajectory is formed by placing the center of this hypervolume along the integrated trajectories of the state variables. The vectors that span the hypervolume will expand or contract along the directions of the state variables according to the local changes in the state variables themselves. The matrix that contains the local changes in the state variables is the Jacobian matrix discussed in Section 1.3.

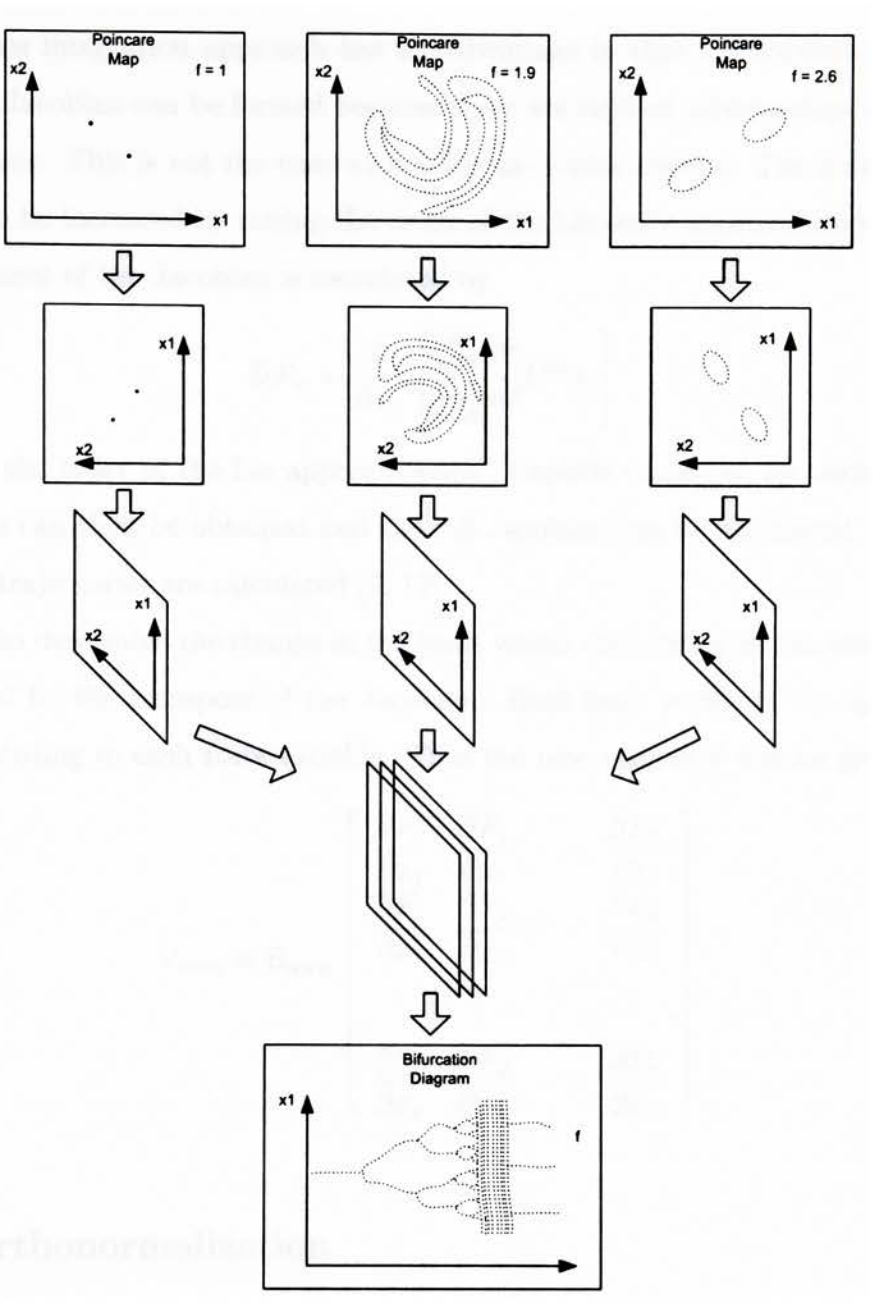


Figure 2.2: The process of creating a Poincaré map bifurcation diagram

2.4.1 Calculating the Jacobian

Typically, the Jacobian is obtained by applying the variational operator to Equation 2.4. The resulting linear equation must then be integrated along with the state equations to calculate the exponents, thus increasing the order and complexity of the integration scheme. It also produces less accurate results because the Jacobian is linear in nature.

The Lie series integration approach has an advantage in that the explicit relationship defining the Jacobian can be formed because there are explicit relationships defining the state equations. This is not the case with a Runge–Kutta scheme. The accuracy of the Jacobian can be increased by raising the order of the Lie series approximation. Thus the (i, j)-th element of the Jacobian is calculated by

$$DF_{ij} = \frac{\partial}{\partial x_j}\left[\sum_{m=0}^{M} \frac{t^m}{m!} U^m x_i\right] \tag{2.22}$$

where M is the order of the Lie approximation. Explicit equations for each element of the Jacobian can then be obtained and the full Jacobian can be calculated at the same time as the trajectories are calculated [7, 12].

In order to determine the change in the basis vector comprising B, the matrix B must be multiplied by the transpose of the Jacobian. Each basis vector of the space will be changed according to each state variable. Thus the new volume V will be given by

$$\mathsf{V}_{n\times n} = \mathsf{B}_{n\times n}\begin{bmatrix} \frac{\partial F_1}{\partial x_1} & \frac{\partial F_2}{\partial x_1} & \cdots & \frac{\partial F_n}{\partial x_1} \\ \frac{\partial F_1}{\partial x_2} & \frac{\partial F_2}{\partial x_2} & \cdots & \frac{\partial F_n}{\partial x_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial F_1}{\partial x_n} & \frac{\partial F_2}{\partial x_n} & \cdots & \frac{\partial F_n}{\partial x_n} \end{bmatrix} \tag{2.23}$$

2.4.2 Orthonormalization

A chaotic attractor acts not only to stretch the state–space, but folds it as well. Since Lyapunov exponents only measure the stretching behavior, it is necessary to normalize the new basis vectors, or the rows of V, to keep the magnitudes small enough that they are not subjected to the folding characteristics of the chaotic attractor. Also, recall from Section 1.3.5 that the behavior of the hypervolume will be dominated by the largest Lyapunov exponent. This means that successive multiplications of the Jacobian matrix will make one of the magnitude of the vectors very large compared to the others. If the magnitudes of these other vectors tend to zero, this would result in an ill–conditioned

V matrix and undefined Lyapunov exponents. To avoid this, the vectors must also be orthogonalized. A method for achieving both normalization and orthogonalization of vectors is Gram–Schmidt orthonormalization. A set of vectors $\vec{v}_i$ is transformed into an orthonormal set $\vec{u}_i$ that spans the same vector space as $\vec{v}_i$. For a n–dimensional system, the process is as follows.

$$\begin{aligned}
\vec{v}_1 &= \delta\vec{x}_1 \\
\vec{u}_1 &= \frac{\vec{v}_1}{\|\vec{v}_1\|} \\
\vec{v}_2 &= \delta\vec{x}_2 - \langle\delta\vec{x}_2, \vec{u}_1\rangle\vec{u}_1 \\
\vec{u}_2 &= \frac{\vec{v}_2}{\|\vec{v}_2\|} \\
&\vdots \\
\vec{v}_n &= \delta\vec{x}_n - \langle\delta\vec{x}_n, \vec{u}_1\rangle\vec{u}_1 - \cdots - \langle\delta\vec{x}_n, \vec{u}_{n-1}\rangle\vec{u}_{n-1} \\
\vec{u}_n &= \frac{\vec{v}_n}{\|\vec{v}_n\|}
\end{aligned} \tag{2.24}$$

In the above equations, $\|\cdot\|$ is the Euclidean norm and $\langle\cdot,\cdot\rangle$ is the inner product.

2.4.3 Approximation

Lyapunov exponents vary slightly across the phase space, so typically they are calculated as average expansion or contraction rates. For a K number of calculations with a time step size T, the Lyapunov exponents λ_i are given by

$$\lambda_i \approx \frac{1}{KT}\sum_{k=1}^{K} \ln \|\vec{v}_i^{(k)}\| \tag{2.25}$$

The orthonormalized V matrix is then used as the B matrix in the next iteration step. For linear systems, Lyapunov exponents are the real parts of the eigenvalues. Recall that a linear systems analysis assumes a solution of the form $\vec{x} = \vec{v}e^{\lambda t}$. Thus the same procedure can be used to calculate the Lyapunov exponents for linear systems. For nonlinear systems, the approach is typically modified to use a logarithmic base of 2 in Equation 2.25. The units of the Lyapunov exponents are then bits/sec and represent the rate at which numerical precision is lost.

$$\lambda_i \approx \frac{1}{KT}\sum_{k=1}^{K} \frac{\log \|\vec{v}_i^{(k)}\|}{\log 2} \tag{2.26}$$

2.4.4 Classification of systems

Lyapunov exponents are very useful in classifying the steady–state behavior for systems containing attractors. If a solution is attracting, by definition, the total contraction rate must exceed the total expansion rate. So, the sum of the Lyapunov exponents must always be less than zero.

$$\sum_{i=1}^{n} \lambda_i < 0 \tag{2.27}$$

Non–chaotic attractors can be classified as follows [9].

- For a stable fixed point, $\lambda_i < 0$ for $i = 1, \ldots, n$
- For a stable limit cycle, $\lambda_1 = 0$, and $\lambda_i < 0$ for $i = 2, \ldots, n$
- For a stable two–torus, $\lambda_1 = \lambda_2 = 0$, and $\lambda_i < 0$ for $i = 3, \ldots, n$
- For a stable K–torus, $\lambda_1 = \cdots = \lambda_K = 0$, and $\lambda_i < 0$ for $i = K+1, \ldots, n$

It was previously mentioned that chaotic systems have at least one positive Lyapunov exponent. The list shown above indicates that any attracting solution other than a fixed point must have one zero exponent. With the condition in Equation 2.27, it must be concluded that a three–dimensional chaotic system will have exponents of the form $(+, 0, -)$. That is, $\lambda_1 > 0$, $\lambda_2 = 0$, and $\lambda_3 < 0$. Another condition imposed by Equation 2.27 is $\|\lambda_3\| > \|\lambda_1\|$. It follows from these three conditions that chaos cannot occur in a first or second–order autonomous continuous time system, or in a first–order non–autonomous continuous time system. Thus chaos cannot exist in the phase plane.

For a fourth–order system, three possibilities exist for the signs of the Lyapunov exponents [9].

- $(+, 0, -, -)$: $\lambda_1 > 0$, $\lambda_2 = 0$, and $\lambda_4 \leq \lambda_3 < 0$.
- $(+, +, 0, -)$: $\lambda_1 > 0 \leq \lambda_2 > 0$, and $\lambda_3 = 0$ and $\lambda_4 < 0$. This is known as *hyper–chaos.*
- $(+, 0, 0, -)$: $\lambda_1 > 0$, $\lambda_2 = \lambda_3 = 0$, and $\lambda_4 < 0$. This corresponds to a chaotic two–torus.

This classification can be extended for higher order systems.

2.4.5 Lyapunov Spectrum

Similar to a Poincaré map bifurcation diagram, a Lyapunov spectrum shows the change in Lyapunov exponents against a varying control parameter. A system is integrated until the Lyapunov exponents converge. The control parameter is then incremented and the system is integrated again until the Lyapunov exponents converge. The process is repeated throughout the entire range of the control parameter. The trends of individual converged Lyapunov exponents can then be plotted against the control parameter. When an exponent trend rises above the abscissa, it indicates the onset of chaotic behavior.

It is now evident that at the two separate analysis tools of Poincaré maps and Lyapunov exponents can confirm any behavior outlined in Chapter 1. A third method is discussed in the next section.

2.5 Power Spectral Density

The power spectral density (PSD) represents the energy associated with a signal as a function of frequency. Mathematically, it can be obtained from the Fourier transform of the autocorrelation of the signal being analyzed. For a windowed data set, an estimate of the PSD is given by

$$\hat{\mathcal{P}}_{gg}[k] = \frac{1}{CN}|\Gamma[k]|^2 \tag{2.28}$$

where $\Gamma[k]$ is the R-point discrete Fourier transform (DFT) of the length-N data sequence, and C is a normalization factor based on the window (see [13] for details). $\hat{\mathcal{P}}_{gg}[k]$ is known as a periodogram if the window is rectangular, and a modified periodogram for other window types. Typically, a smoother power spectrum can be obtained by calculating multiple periodograms and averaging. The full data set is divided into K overlapping regions. These regions are then windowed and the periodograms are calculated for each region. The periodograms are then averaged together. The full method is outlined in [14]. In this method, the DFT is calculated using fast Fourier transform algorithms.

Periodic solutions of the nonlinear differential equations will be represented at sharp spikes in the PSD. When a period–doubling bifurcation occurs, new magnitude peaks will appear at subharmonic multiples of the forcing frequency. Quasiperiodicity will be represented by distinct frequency components, but unlike period–2 or period–4 solutions, the frequencies will not be multiples of each other. A chaotic signal will be represented by a broad range of frequency components that is typically associated with a random signal or noise as shown in Figure 2.3.

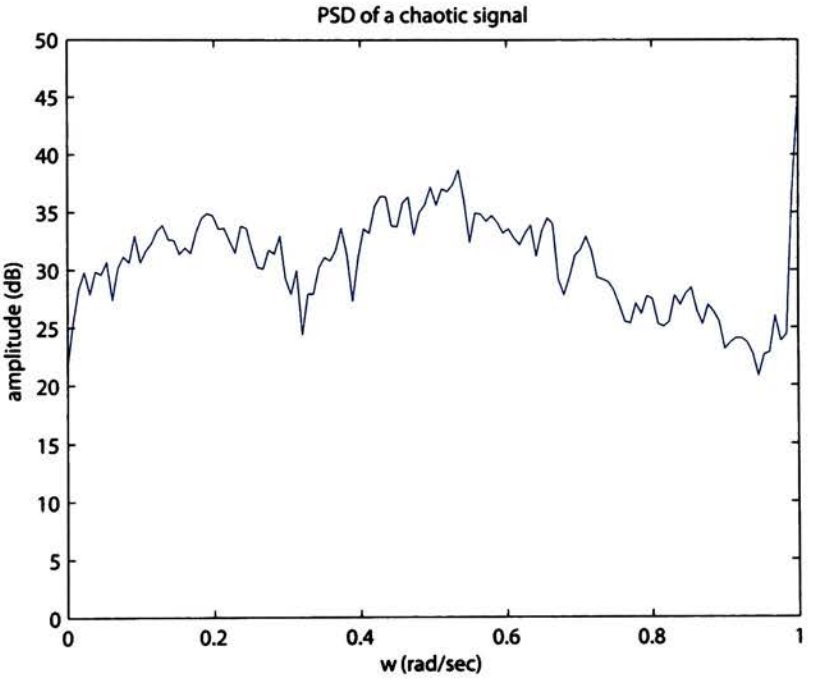


Figure 2.3: The PSD of a chaotic signal

Chapter 3

The Duffing System

The single Duffing oscillator system is the foundation of more complex multiple oscillator systems. Much like a vibrations student learns the nuances of single linear oscillator systems before studying the more complex coupled systems, a nonlinear investigator must first understand the behaviors that can result from a single nonlinear oscillator. A brief introduction to previous research in single Duffing systems is presented before the previous research on the more complicated linear and nonlinear coupled multiple oscillator systems.

The Duffing System or Duffing's equation is one of the most widely known nonlinear systems. It is a standard second–order forced differential equation with the addition of a cubic restoring term.

$$\ddot{x} + \gamma\dot{x} + \alpha x + \beta x^3 = f(t) \tag{3.1}$$

Depending on the choices of the parameters γ, α, and β, and the choice of the forcing function $f(t)$, this system can model a variety of systems, some of which are described in this chapter. The potential function of the system is defined as

$$V(x) = \frac{\alpha}{2}x^2 + \frac{\beta}{4}x^4 \tag{3.2}$$

With $\alpha < 0$ and $\beta > 0$, this is often called a double–well potential for the two symmetric wells on either side of the origin (see Figure 3.1).

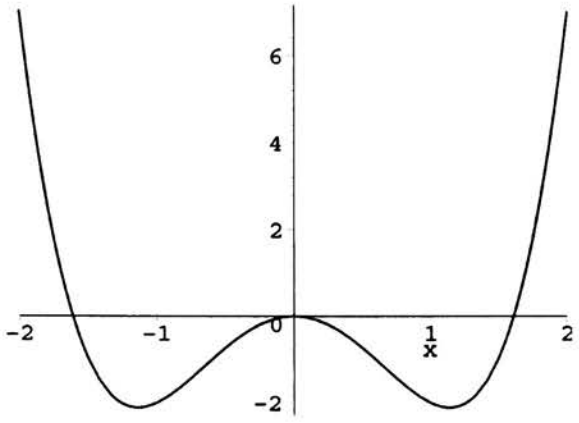


Figure 3.1: The double–well potential in Duffing's equation ($\alpha < 0$, $\beta > 0$)

3.1 Single Duffing Oscillator Systems

3.1.1 A Magneto–Elastic Duffing System

A version of Equation 3.1 has been studied very extensively by a group of researchers at Cornell University, most notably Francis Moon and Philip Holmes. Equation 3.1 with $f(t) = f\cos\omega t$ can be considered to be a Galerkin approximation of a sinusoidally forced beam vibrating in a single mode in the presence of an external magnetic field. In this case, x represents the amplitude of the mode. The magnetic field under consideration is typically strong enough resulting in beam buckling, thus creating multiple fixed points. Using the apparatus shown in Figure 3.2, along with analog and digital computer simulations, and the methods outline in Chapter 2, Moon and Holmes have verified both experimentally and theoretically the existence many types of nonlinear dynamic behavior. The Poincaré map in Figure 2.1(a) was a product of their research. For more information

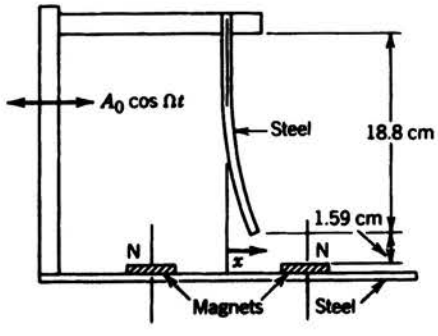


Figure 3.2: Experimental apparatus for magneto–elastic Duffing system [2]

and results regarding the magnetically buckled beam experiments, the reader is referred to the articles cited in [2, 15, 16].

3.1.2 The Japanese Attractor

Another important researcher of Duffing's equations is Yoshisuke Ueda of the Kyoto University in Japan. He has studied the equation in the following form

$$\ddot{x} + k\dot{x} + x^3 = B\cos t \tag{3.3}$$

where x represents the magnetic flux in a nonlinear inductor. The equation has been non–dimensionalized such that the entire dynamics of his circuit could be realized by varying the parameters k and B, and the initial conditions $x(0)$ and $\dot{x}(0)$. The parameter k represents a non–dimensional damping term and B is a non–dimensionalized forcing amplitude. Ueda was a major proponent of the idea of chaotic behavior. Prior to his research in the early 1970's, chaotic behavior was hardly reported in the literature, despite the fact that identical systems had been previously studied for decades [2]. Most often, any non–quasiperiodic behavior was reported as consisting of "irregular vibrations", turbulence, or simply as noise. The Poincaré map in Figure 2.1(b) was a product of his research. A book containing the majority of his research, including some English translations of articles, is cited in the bibliography [10].

3.2 Coupled Duffing Oscillator Systems

More recently, the focus on nonlinear dynamics has been extended to coupled systems. Coupled oscillator systems can describe a variety of physical phenomena in many branches of engineering, physics, and biology.

3.2.1 Linear Coupling

A bifurcation analysis of two linearly coupled Duffing oscillators was performed by Kozlowski, Parlitz, and Lauterborn [17]. The system of equations studied is shown as

$$
\begin{aligned}
\dot{y}_1 &= y_2 \\
\dot{y}_2 &= -dy_2 - y_1 - y_1^3 + c(y_3 - y_1) + f\cos 2\pi y_5 \\
\dot{y}_3 &= y_4 \\
\dot{y}_4 &= -dy_4 - y_3 - y_3^3 - c(y_3 - y_1) \\
\dot{y}_5 &= \omega/2\pi
\end{aligned}
\tag{3.4}
$$

The forcing amplitude f and the forcing frequency ω were chosen as the control parameters for the system. The damping parameter d, and the linear coupling strength c were fixed throughout the course of the study. Typically the forcing frequency was varied with a step from 0.01 to 0.0001 while the forcing amplitude was held constant. Poincaré map derived bifurcation diagrams were given for the first two Poincaré variables u_1 and u_2 corresponding to y_1 and y_2 in the state equations. They found that the common period–doubling bifurcations and route to chaos found in a single Duffing oscillator persists in the system they studied. Additionally, quasiperiodicity and its associated route to chaos are now observed for this system due to Hopf bifurcations which cannot exist in the single oscillator system. The Hopf bifurcations were observed for period–2, 3–tori, and chaotic attractors.

Another bifurcation analysis by Stagliano, Wersinger, and Slaminka focused on the period–doubling bifurcations of tori [18]. The system they studied was a quasiperiodically forced coupled Duffing system. In this system the coupling term is additive.

$$
\begin{aligned}
\ddot{x}_1(t) &= -\gamma_1\dot{x}_1(t) + \frac{1}{2}\sigma_1 x_1(t)\left[1 - x_1^2(t)\right] + f_1\cos(\omega_1 t) + \epsilon_1\left[x_1(t) + x_2(t)\right] \\
\ddot{x}_2(t) &= -\gamma_2\dot{x}_2(t) + \frac{1}{2}\sigma_2 x_2(t)\left[1 - x_2^2(t)\right] + f_2\cos(\omega_2 t) + \epsilon_2\left[x_2(t) + x_1(t)\right]
\end{aligned}
\tag{3.5}
$$

Here $x_i(t)$ describes the displacement amplitude, γ_i is the damping, σ_i is the restoring force strength, f_i is the forcing amplitude, ω_i is the forcing frequency, and ϵ_i is the coupling strength of each oscillator i. A torus initially exists when the relation ω_1/ω_2 is irrational, and each oscillator could independently be considered to have a stable limit cycle. Period–doubling of tori can occur if one of the above equations undergoes a period–doubling while the other remains the same. Intuitively it was thought that the period–doubling of tori could be a route to chaos, much like the well–known period–doubling

of limit cycles. However, this was not the case. Period–doubling in tori is interrupted when the torus becomes wrinkled and then discontinuous. The authors hypothesized that period–doubling could still occur in these *destroyed* tori.

The system parameters γ_i and σ_i, along with the coupling strength ϵ_i were considered to be equal between the two oscillators. The forcing on the second oscillator was considered constant in terms of amplitude and frequency. The two bifurcation parameters were chosen as f_1 and ϵ. Thus the contributions of the second oscillator on the first were constant, and all of the interesting behavior could be captured by observing the first oscillator. The period–doubling behavior was controlled by the parameter f_1 and that the destruction of the torus was governed by ϵ. The period–doubling sequence was stopped at period–2 for a stable torus. Similar period–doubling occurred in an initially destroyed torus as well. These behaviors were captured in three–dimensional Poincaré maps and power spectra. The phenomena was studied in further detail in a simple three–dimensional discrete map.

Yin, Dai, and Zhang investigated the phase difference and frequency detuning in two linearly coupled Duffing oscillators [19]. The system of equations studied was

$$\begin{aligned} \dot{x} &= y + C(u - x) \\ \dot{y} &= -\alpha y + x - x^3 + \beta \cos \omega t \\ \dot{u} &= v + C(x - u) \\ \dot{v} &= -\alpha v + u - u^3 + \beta \cos\left[(\omega + \Delta\omega)t + \varphi\right] \end{aligned} \tag{3.6}$$

Here φ and C are the phase difference and coupling strength respectively, and $\Delta\omega$ represents a detuning in the frequency between the two oscillators. The authors investigated the phase difference and detuning independently.

The phase difference was found to prevent synchronization between the two oscillators with a small coupling strength. For a sufficient phase difference, chaotic behavior is avoided and the motion is periodic. This result was confirmed through the use Poincaré maps and Lyapunov exponents. For larger coupling strength, the effects of the phase difference was suppressed. For these cases, the coupling strength caused the two oscillators to remain synchronized.

A small detuning frequency with a small coupling strength results in two oscillators

that show almost no correlation. Increased coupling strength results in what the authors call *stochastic breathing*, bifurcation delay, and stochastic transition. The breathing results from alternating periodic motion and chaotic motion. Other authors refer to this as intermittent chaos [1, 2]. The bifurcation delay results when the chaotic behavior does not transition to periodic motion immediately. A stochastic transition occurs breaking the symmetry of the Poincaré map. The authors attribute the transition to an effective time dependent phase difference due to the nature of the detuning. A phase difference of $\varphi(t) = \Delta\omega t$ results from the detuning parameter $\Delta\omega$.

Musielak, Musielak, and Benner expounded upon earlier doctoral research performed by Benner in their studies of systems of coupled Duffing oscillators with multiple degrees of freedom [20, 21]. Their studies were not limited to two oscillators, but examined as many at ten coupled oscillator systems. Each oscillator was coupled to at most two other oscillators. A schematic representation of the three degree of freedom coupled oscillator system is shown in Figure 3.3.

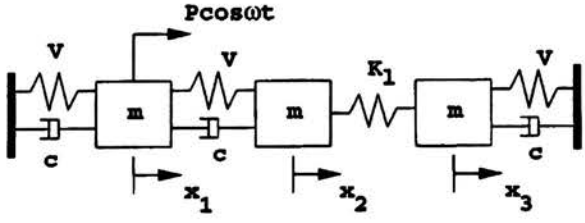


Figure 3.3: Three coupled oscillator system studied in [20, 21]

The equations of motion for the system are

$$\begin{aligned} m\ddot{x}_1 + Vx_1^3 + c\dot{x}_1 - V(x_2 - x_1)^3 - c(\dot{x}_2 - \dot{x}_1) &= P\cos\omega t \\ m\ddot{x}_2 + V(x_2 - x_1)^3 + c(\dot{x}_2 - \dot{x}_1)^3 - K_l(x_3 - x_2) &= 0 \\ m\ddot{x}_3 + Vx_3^3 + c\dot{x}_3 + K_l(x_3 - x_2) &= 0 \end{aligned} \tag{3.7}$$

In the above equations m is the mass, V is the cubic stiffness, K_l is the linear coupling stiffness, c is a viscous damping coefficient, P is the amplitude of the forcing function, and ω is the frequency. The system of equations was then converted to a seventh–order autonomous system.

$$
\begin{aligned}
\dot{y}_1 &= y_2 \\
\dot{y}_3 &= \left[B\cos y_3 - ky_2 - y_1^3 + (y_4 - y_1)^3 + k(y_5 - y_2)\right] \\
\dot{y}_3 &= \omega \\
\dot{y}_4 &= y_5 \\
\dot{y}_6 &= \left[k_c(y_6 - y_4) - k(y_5 - y_2) - (y_4 - y_1)^3\right] \\
\dot{y}_6 &= y_7 \\
\dot{y}_7 &= \left[-k_c(y_6 - y_4) - y_6^3 - ky_7\right]
\end{aligned}
\tag{3.8}
$$

The forcing amplitude B was the control parameter. This process was extended for the other coupled systems. Systems of an odd number of oscillators were asymmetric with respect to the linear spring while systems with an even number were symmetric about the linear spring.

The numerical methods utilized were Lyapunov exponents, Poincaré maps, and power spectra. The systems of equations were integrated using a fourth–order Runge–Kutta scheme with 1600 forcing periods discarded as the transient solution before the Poincaré maps were calculated. The forcing amplitude was varied from 0 to 150. Full numerical modeling details can be found in [21]. The period–doubling route to chaos was prevalent in systems of lower order. The authors also observed what they called *subharmonic smearing* of points on the Poincaré map before the onset of chaos. Typically these points smeared into the shape of the chaotic attractor. Quasiperiodicity was also present throughout many of the studied systems. As the number of oscillators increased, regions of chaotic behavior became more limited and less frequent. Also the period–doubling route to chaos was replaced by what the authors call *crisis*, which is a sudden onset of chaos.

Different conditions were placed on the two coupled system. The authors investigated effects of the coupling strength k_c from 10 to 1000. It was found that increased coupling strength did not change the routes to chaos, but there was an increase in the chaotic regions. An additional sinusoidal forcing term was placed on the second mass. The phase difference between the two forcing functions was varied from 90^o to 270^o. The frequency ratios were also varied from 0.2 to 0.7 with the phasing held constant. The resulting behavior of these alterations to the initial system shows no chaotic motion. The

authors attribute this to mode–locking effects.

The order of the linear coupling element was also increased to a cubic. In the single forced systems and the doubly forced system with the above phasing, period–doubling and crisis were the routes to chaos. When the phasing was set to 180^o, crisis events dominate and the number of chaotic regions increased dramatically. For the varying frequency ratios, crisis led to regions of very low intensity chaos with the maximum Lyapunov exponent not exceeding 0.08 bits/sec. No quasiperiodic motion was exhibited by any of the two coupled systems under any conditions.

3.2.2 Nonlinear Coupling

Nonlinearly coupled Duffing oscillators were studied earlier than the linearly coupled versions. The classical problem of a vibrating string has a well–known linearized solution. However, this solution is only valid when the initial tension and the displacement of the string are such that there is minimal variation in the tension. If these conditions are not met, then the problem is necessarily nonlinear in nature due to the coupling of the transverse and longitudinal vibration modes. This problem has practical applications in musical instrument design both in terms of the design of bridge structures, electric pickups, and in the case of pianos, the striking method [22, 23, 24]. Many of these early authors focused on perturbation methods and Fourier analysis to derive equations of motion of the fundamental mode of the string. Methods were typically oriented to find frequency ranges in which planar and circular or "whirling" motions exist, and the paths of those circular motions [25, 26, 27, 28]. Miles, Anand, and Narisimha all separately derived the exact form of a conservative two coupled Duffing oscillator system. The Miles form is given by

$$\left[\delta^{-1}\left(D^2+1\right)+\frac{2}{3}\left(\alpha^2+\beta^2\right)\right]\{\alpha,\beta\}=\{1,0\}\cos\omega t \tag{3.9}$$

where D is a differential operator. This is simply two conservative Duffing oscillators coupled through a cubic $\alpha\beta^2$ or $\alpha^2\beta$ term depending on the oscillator. The parameters α and β represent the Fourier coefficients of the displacements, or the displacement amplitudes of the fundamental modes. Refer to [26] for the exact meaning of the other

variables. There is no mention of chaos in these early articles.

More recently, Elliot studied the effects of a freely vibrating string [24, 29]. The purpose of the studies was to describe the parametric excitation through nonlinear effects intrinsic to the geometry of the string. It was also shown that energy transfer exists between the x and y perpendicular modes. In this case, strings should properly be considered as two parametrically coupled oscillators. The potential energy of the string was derived based on the mechanical work performed by the stretching of the string. The kinetic energy was derived as simply as the mass multiplied by the sum of the square of each of the velocity components. In non–dimensional form, this resulted in

$$\begin{aligned} \ddot{X} + \omega^2 X \left[1 + \sigma \left(X^2 + Y^2\right)\right] &= 0 \\ \ddot{Y} + \omega^2 Y \left[1 + \sigma \left(X^2 + Y^2\right)\right] &= 0 \end{aligned} \tag{3.10}$$

Assuming that one of the perpendicular modes will dominate, i. e. $Y << X$, resulted in a modification of each equation.

$$\begin{aligned} \ddot{X} + \omega^2 X + \sigma \omega^2 X^3 &= 0 \\ \ddot{Y} + \omega^2 Y \left(1 + \sigma X^2\right) &= 0 \end{aligned} \tag{3.11}$$

The first equation can be solved using standard perturbation methods. This resulted in equations that describe sinusoidal oscillations at a shifted frequency dependent on the equation parameters and the oscillation amplitude. Additional third harmonic terms were also present in the solution. Substituting the X solution into the second modified equation results in a Mathieu type equation. The simplest exact solutions to the first set of equations are circular motions at a detuned frequency. This system is mathematically similar to the spherical pendulum described by Olsson [30].

Observations of real strings show that the motion typically follows an elliptic path that precesses about its axis of rotation. The original Lagrangian of the system is thus transformed into a rotating frame of reference. Solutions are obtained for the frequency shift in the rotating frame as well as the precession frequency. For circular motions, the equations obtained are in agreement with the exact solution derived earlier in the article.

It is now obvious that strictly planar motion of the wire will remain planar. However, the slightest deviation from planar motion will result in precessing elliptical orbits, and

subsequent transfer of energy between the perpendicular modes.

It is often the case in musical instruments that the symmetry of Equation 3.10 would be broken through the design of the supporting string bridge. It is difficult to include these effects analytically, but two limit cases can be defined practically. If the quantity of the frequency of the y motion subtracted from the x frequency is much less than the precession frequency, then the effects of the asymmetry are barely noticed due to averaging. The precessional motion will still dominate but at a slightly oscillating frequency due to the change in area of the elliptical orbital path. If the quantity is much larger than the precession frequency, it will appear as though the ellipse is rocking back and forth between the x and y directions. In effect, the value of the precession frequency is changing sign.

In his second article, Elliot considered the effects of forcing in one degree of freedom. New frequency shifts were observed as the solution is obtained from Duffing's harmonic balance method. A relationship between the elliptic amplitudes was also derived. The jump phenomena is also derived and shown for the zero damping case.

Tufillaro was perhaps the first to consider chaos in string vibrations [31]. He studied the same type of string motion as the previous authors. However, his model was slightly different in that it consisted of a mass–less spring with ends that are assumed to be fixed and symmetric. A concentrated mass was placed at the center of the string. The mass can be subjected to both damping and forcing. The model had similar coupling between the transverse directions of the string both directly and indirectly through the longitudinal vibrations of the string. Tufillaro derived an equation for the restoring force of the "spring". From that, the final system equations of motion is given as a two-dimensional conservative cubic oscillator. Forcing and damping are added into the equation to yield

$$\ddot{\vec{r}} + \lambda\dot{\vec{r}} + \omega_0^2\left(1 + K\vec{r}^2\right)\vec{r} = \vec{f}(\omega t) \tag{3.12}$$

The equations were non–dimensionalized by the natural frequency and string length resulting in

$$\vec{s}'' + \beta\vec{s}' + \left(1 + \alpha\vec{s}^2\right)\vec{s} = \vec{g}(\gamma\tau) \tag{3.13}$$

Exact solutions for circular and planar motion were developed for the undamped, unforced case. Circular motion was found to be a solution if the frequency was detuned. The solution for planar motion involves elliptic integrals, and again shows that the frequency would be adjusted to a new value.

Restricting motion to a single plane, but including both forcing and damping, results in a Duffing equation. A single Duffing equation is complicated enough to exhibit multiple periodic solutions, quasiperiodic orbits, and chaotic motion. Also, hysteresis effects cause discontinuous transitions between separate stable solutions. These transitions are also known as jump phenomena.

A linear approximation to the full equation without forcing results in elliptical motion. Similarly to Elliot, it was assumed that the solution to the equations could be circular motion in a rotating reference frame. The form of a damped circular motion with first and third Fourier components was assumed as the solution. Similar detuning formulations for angular frequency and precession rate were obtained. However, solutions for forced circular motions were not investigated.

Transitions from periodic to chaotic behavior were identified through the use of Poincaré map bifurcation diagrams. Diagrams were constructed for forced planar motion and forced circular motion. Simulations were run with parameters near resonance, based on a 1% detuning. Forcing was varied between 50 and 55. The first 400 cycles were disregarded assuming that they are part of the transient solution. The attracting solution was sampled once every period of forcing. The asymptotic solution was then used as the initial conditions for the next simulation in which the forcing was incremented by a small amount. Due to the hysteresis effects observed earlier in the article, the forcing was scanned in both directions, i. e. from 50 to 55, and from 55 to 50. In the diagrams, orbits appear to jump between attractors as the forcing amplitude is varied. This was attributed to an artifact of the step size of 0.1 as the forcing increment. The basins of attraction are intertwined in such a way that this small step size would cause the orbit to jump to another attractor. Simulations also showed that a transition from planar to circular motion takes place with a forcing amplitude of 61.7. The diagrams were not meant to be a comprehensive study for the dynamics of a vibrating string, but merely to show

that it can exhibit complex motions under realizable experimental operating conditions.

In their investigation of nonlinear interactions between widely spaced vibration modes, Nayfeh and Nayfeh used a two coupled Duffing oscillator system for their modeling equations [32]. The authors' primary focus was determining if low–amplitude excitation near a higher frequency mode could in turn cause large–amplitude vibrations of the low–frequency fundamental vibration mode. Large–amplitude low–frequency vibrations can be particularly dangerous in engineering applications. The system equations studied were as follows

$$\begin{aligned} \ddot{u}_1 + 2\epsilon\mu_1\dot{u}_1 &= -\epsilon^2\left(4\alpha_1 u_1^3 + \alpha_2 u_1 u_2^2\right) \\ \ddot{u}_2 + 2\epsilon\mu_2\dot{u}_2 &= \epsilon\left(\alpha_3 u_2^3 + \alpha_4 u_1^2 u_2 + f\cos\Omega t\right) \end{aligned} \tag{3.14}$$

where ϵ is the ratio between the linear natural frequencies of the high and low modes and is assumed to be positive and small. The high frequency mode has coordinate u_2. Its frequency has been non–dimensionalized to unity. The low frequency mode has coordinate u_1 and a linear natural frequency of ϵ. The system has viscous damping and the coefficients are represented by μ_1 and μ_2. The cubic nonlinearities have coefficients α_i. External forcing is applied to the high frequency mode near its natural frequency.

Averaging was performed on the equations such that the high frequency oscillations of the higher mode would be smoothed so as to not affect the low mode. Typically, the response of each vibration mode is of the form $u_i = a_i(t)\cos\left(\omega_i t + \beta_i(t)\right)$, where $a_i(t)$ and $\beta_i(t)$ are slowly varying. The original equations are then manipulated into first–order differential equations for $a_i(t)$ and $\beta_i(t)$, and integrated with respect to time over the oscillation period. In this sense, $a_i(t)$ and $\beta_i(t)$ are assumed to be constant because they change very little over one oscillation period. It was assumed that the time–varying amplitude and phase of the high mode oscillator and u_1 are slowly varying. This resulted in the averaged or modulation equations

$$
\begin{aligned}
\dot{u}_1 &= \epsilon v_1 \\
\dot{v}_1 &= -\epsilon \left(u_1 + 2\mu_1 v_1 + 4\alpha_1 u_1^3 + \frac{1}{2}\alpha_2 u_1 a^2 \right) \\
\dot{a} &= -\epsilon \left(\mu_2 a + \frac{1}{2} f \sin\beta \right) \\
\dot{\beta} &= -\epsilon \left(\frac{1}{2}\sigma + \frac{1}{2}\alpha_4 u_1^2 + \frac{3}{8}\alpha_3 a^2 + \frac{f}{2a}\cos\beta \right)
\end{aligned}
\tag{3.15}
$$

Since ϵ appears as a common factor in each equation, the character of motion will not depend on this parameter. Fixed points of the system and their corresponding stabilities were analyzed. By analyzing the eigenvalues of the Jacobian, it was possible to detect saddle–node and Hopf bifurcations.

Frequency response curves showing stable and unstable oscillations were calculated for certain parameter values. The stability transitions were then investigated and critical values of some of the equation parameters were identified. For certain conditions, increasing the amount of damping caused undesirable oscillations and loss of stability. Finally, numerical simulations were performed with the averaged and original equations to confirm the stability characteristics identified in the frequency response curves, and to verify the averaging method. The numerical studies identified the existence of both Hopf and period–doubling bifurcations and the merging of chaotic attractors. The averaged equations were shown to be in good agreement with the original equations for small values of the parameter ϵ.

The study by Nabergoj, Tondl, and Virag focused on autoparametric resonance in a two coupled Duffing oscillator system [33]. Similarly to Nayfeh and Nayfeh, they were concerned with the conditions in which an externally excited subsystem could cause significant oscillations in a non–excited coupled subsystem. The potential energy of the system they studied was

$$
U(x,y) = \frac{1}{2}\left(k_1 x^2 + k_2 y^2\right) + \frac{1}{4}k_0\left(x^2 + y^2\right)^2 \tag{3.16}
$$

which corresponds to the following equations of motion.

$$\begin{aligned} m\ddot{x} + b_1\dot{x} + k_1 x + k_0\left(x^2 + y^2\right)x &= f\cos\omega t \\ m\ddot{y} + b_2\dot{y} + k_2 y + k_0\left(x^2 + y^2\right)y &= 0 \end{aligned} \tag{3.17}$$

The first equation refers to the excited subsystem and the second to the non–excited subsystem. It is evident that the second equation has a trivial solution of $y(t) = 0$. The authors refer to a non–trivial solution in x and the trivial solution in y as the semi–trivial solution. Frequency response curves and stability regions for this system were identified. These stability regions coincided exactly to the regions in Duffing's equations. Equation 3.17 reduces to a Duffing system when $y(t) = 0$. The regions in which the semi–trivial solution is unstable are the regions in which autoparametric resonance is initiated. The non–trivial solution in which both subsystems are oscillating was investigated next. New frequency response curves and stability regions for each oscillator were identified.

A domain of attraction study was also performed to determine the boundaries of the semi–trivial and non–trivial solutions. Normally, basins of attraction are defined by varying the initial conditions of the systems of equations and determining the range over which certain solutions are attracting. However, for systems of higher order this becomes prohibitive due to the impossibility of geometric representation. Previously, one of the authors identified an alternative method to perform this type of study. The equations of motion are solved for initial conditions which result in a stable solution with no transient. At a specified time, a disturbance is applied and the resulting motion is observed. If the disturbance can be characterized by two parameters, then the results of this study can be shown on a plane. In this case, a sinusoidal pulse was the disturbance. The time duration and amplitude of the pulse were varied as the control parameters. A grid was placed over this parameter space where each grid cell was a different pulse corresponding to the varying amplitude and pulse time. Transitions to a different solution were shown by shaded cells, whereas cells that did not transition to a different solution remained white. These transitions are characterized as semi–trivial/semi–trivial, semi–trivial/non–trivial, non–trivial/non–trivial, and non–trivial/semi–trivial corresponding to the solutions presented earlier in the article. Probability distributions of the occurrence in which the disturbed solution tends to the initial solution were also calculated.

Mahmoud, Mohamed, and Aly investigated a complex form of a Duffing oscillator system [34].

$$\ddot{z} - z + \alpha\dot{z} + \epsilon z|z|^2 = \gamma' \tag{3.18}$$

$$\ddot{z} - kz + \epsilon z|z|^2 = \gamma \exp(\omega t) \tag{3.19}$$

where $\gamma' = \sqrt{2}\gamma\exp(i\pi/4)$; γ, α, k, and ω are positive, and $z = x + iy$ is complex. Equation 3.18 is a coupled Duffing oscillator of the form

$$\begin{aligned} \ddot{x} - x + \alpha\dot{x} + \epsilon x\left(x^2 + y^2\right) &= \gamma\cos\omega t \\ \ddot{y} - y + \alpha\dot{y} + \epsilon y\left(x^2 + y^2\right) &= \gamma\cos\omega t \end{aligned} \tag{3.20}$$

and Equation 3.19 is of the form

$$\begin{aligned} \ddot{x} - kx + \epsilon x\left(x^2 + y^2\right) &= \gamma\cos\omega t \\ \ddot{y} - ky + \epsilon y\left(x^2 + y^2\right) &= \gamma\sin\omega t \end{aligned} \tag{3.21}$$

The first set of equations are symmetric in nature while the second set are not. The first set was studied in this article.

Fixed points and their stabilities were calculated for the unforced case. The system has three fixed points, one of which is a saddle–node. The other two fixed points are either sinks, or star and stable nodes depending on the parameter α. The stability of these fixed points was verified via Poincaré maps. In the unforced case it was found that for low forcing amplitudes, the sinks became attracting orbits. The existence of a strange attractor for certain conditions was also verified through a Poincaré map. Lyapunov exponents were calculated using the traditional method of integrating the variational equation along with the systems of equations. The fourth order Runge–Kutta method was used as the integration scheme. The existence of the chaotic attractor found in the previous section was confirmed by a positive maximal Lyapunov exponent. The power spectrum of the chaotic time series was also calculated and showed several peaks separated by broad–band domains. A method of chaos control was implemented for both systems with positive results.

Lim, Cartmell, Cardoni, and Lucas have used a coupled Duffing oscillator system as a platform for understanding the vibratory response of ultrasonic cutting equipment

[35]. It is often the case in ultrasonic tooling equipment that components possessing both hardening and softening characteristics are coupled in a serial manner. An hypothesized model (Figure 3.4) has system equations of the form

$$\begin{aligned} m_1\ddot{x}_1 + c_1\dot{x}_1 + c_2(\dot{x}_1 - \dot{x}_2) + k_1x_1 + k_2(x_1 - x_2) + h_1x_1^3 + h_2(x_2 - x_1)^3 &= F\cos\Omega t \\ m_2\ddot{x}_2 + c_2(\dot{x}_2 - \dot{x}_1) + k_2(x_2 - x_1) - h_2(x_2 - x_1)^3 &= 0 \end{aligned} \tag{3.22}$$

where h_1 and h_2 represent nonlinear stiffness coefficients. The classical perturbation

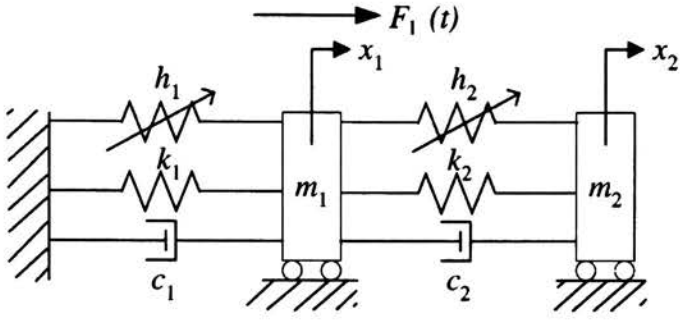


Figure 3.4: Coupled oscillator system studied in [35]

method of multiple scales was used to calculate amplitude for resonance conditions based on h_2. The *Mathematica* software package was used to calculate frequency response curves based on the initial differential equations. These curves exhibited the jump phenomena associated with hysteresis. The *Dynamics 2* software was used to calculate the bifurcations curves with excitation force and frequency as control parameters. Chaotic behavior on these curves was verified using Lyapunov exponents. Experimental work using a variety of configurations of the ultrasonic equipment displayed similar behaviors to that of the hypothesized model.

In a series of articles, Rajasekar and Paul Raj have investigated a variety of behaviors in a two coupled Duffing system [36, 37, 38, 39].

$$\begin{aligned} \ddot{x} &= -d\dot{x} - 2A_1x - 4\alpha_1x^3 - 2\delta xy^2 + f_1\cos\Omega_1 t \\ \ddot{y} &= -d\dot{y} - 2A_2y - 4\alpha_2y^3 - 2\delta x^2y + f_2\cos\Omega_2 t \end{aligned} \tag{3.23}$$

The first article deals with the Painlevé property and the integrability of the equations in the unforced condition. The Painlevé test was performed to identify the limits of integrability of the system. Exact analytical solutions for the integrable cases were obtained.

The effects of periodic forcing were also studied. The dynamics of the system were investigated using the amplitudes of the external forcing function as control parameters. The analysis was performed for three physically interesting potential functions based on the potential function V given by

$$V(x,y) = A_1x^2 + \alpha_1x^4 + A_2y^2 + \alpha_2y^4 + \delta x^2y^2 \tag{3.24}$$

Throughout the analysis it was assumed that A_2 and α_2 had the same signs as A_1 and α_1. The shape of the potential varies with the signs of A_1, α_1, A_2, and α_2. Three cases were investigated.

- Case 1: A_1, α_1, A_2, $\alpha_2 > 0$, single well with infinite height potential
- Case 2: A_1, $A_2 < 0$, α_1, $\alpha_2 > 0$, potential with center hump
- Case 3: A_1, $A_2 > 0$, α_1, $\alpha_2 < 0$, single well with finite height hump potential

For the first case, numerical studies indicated period–doubling bifurcation routes to chaos. The analysis was performed using bifurcation diagrams and the maximal Lyapunov exponent. For the second case and small values of forcing amplitude, two limit cycle orbits occur. Both of these orbits exhibit period–doubling routes to chaotic motion. Each attractor possesses its own basin of attraction, in which the initial conditions define the resulting orbital motion. Crisis events cause the motion to transition from chaos back to period-1 limit cycles. It was also found that these two attractors merge into a single attractor at higher forcing values. The third case shows results similar to the first two in terms of periodic orbits and period doubling bifurcations. In contrast with the second case, the bifurcations occur at different values for forcing between the two oscillators.

The second and third articles dealt with migration control and noise–induced jumps in the coupled Duffing systems.

In the fourth article, the equations of motion were considered to be symmetric, i. e. same forcing amplitude, frequency, damping, and coupling strength. The system was studied under a range of initial conditions in the interval $x(0) \in (-1.2, 1.2)$, $y(0) \in (-0.6, 0.6)$ with $\dot{x} = \dot{y} = 0.2$. It was found that the system has six coexisting chaotic attractors for certain parameter values.

The first two attractors identified are symmetric. That is, the phase portraits in the x and y directions are identical. This is due to the fact that the equations decouple under the symmetric initial conditions and the equations can be treated as two separate identical Duffing oscillators. The second two attractors are found to have non–identical phase portraits for each direction. The fifth attractor appears to have identical phase portraits similar to that of the first two attractors; however, in this case the state variables are completely asynchronized. Synchronization was verified using standard plots of position and velocity in x vs. y manner.

With the existence of several attractors in a system, it is of interest to define the basins of attraction for each. Since the original system is four–dimensional, a single four space representation of the basin of attraction is not possible. The problem was reduced by examining a section of the four–dimensional space, specifically the section defined by $x(0) \in (-1.2, 1.2)$, $y(0) \in (-0.6, 0.6)$ and $\dot{x} = \dot{y} = 0.2$. This section was divided into a grid of 120 x 120 points. Using each point as an initial condition, the system was integrated with a fourth–order Runge–Kutta algorithm with a time step of $2\pi/100$. The first 500 forcing cycled were disregarded as transient portions of the solution. The next 1000 points of the Poincaré map were used to identify the attractor.

The synchronization of the system was studied using a continuous feedback control for chaotic motion.

3.3 The Investigated System

The investigators in the majority of the reviewed literature chose to study the varying dynamics of coupled Duffing oscillator systems through a variable forcing condition. The control parameters in these studies were the forcing frequency and amplitude. An engineer will often reasonably understand the forcing conditions in which the designed system will be placed. It is not uncommon to "tune" a system response by changing the damping and stiffness coefficients in order to meet the required operating conditions. It may also be the case that the parameters governing the differential equation are constants, but uncertain within a certain range. It would be beneficial to know what type of behaviors

to expect over the tolerances on the equation parameters, and to design the system to avoid chaotic and quasiperiodic responses. It can also be used to determine whether or not a linearization of the system is valid to use as an approximation for a plant model for use in linear control systems design.

To this end, the dynamics of a two coupled Duffing oscillator will be investigated by varying the values of the stiffness elements in the system. Consider a potential function of the form

$$V(x,y) = \frac{\alpha}{2}x^2 + \frac{\beta}{4}x^4 + \frac{\gamma}{2}y^2 + \frac{\delta}{4}y^4 + \frac{\epsilon}{2}x^2y^2 \tag{3.25}$$

A mass m is placed within this potential field and is subjected to both viscous damping and external sinusoidal forcing. This results in a systems of equations in the following form

$$\begin{aligned} m\ddot{x} + \zeta\dot{x} + \alpha x + \beta x^3 + \epsilon xy^2 &= F(\omega t) \\ m\ddot{y} + \zeta\dot{y} + \gamma y + \delta y^3 + \epsilon x^2 y &= G(\omega t) \end{aligned} \tag{3.26}$$

Introducing the non–dimensional time variable $\tau = \omega t$ results in

$$\begin{aligned} x'' + zx' + ax + bx^3 + exy^2 &= \frac{1}{m\omega^2}F(\tau) \\ y'' + zy' + cy + dy^3 + ex^2y &= \frac{1}{m\omega^2}G(\tau) \end{aligned} \tag{3.27}$$

where

$$z = \frac{\zeta}{m\omega}, \quad a = \frac{\alpha}{m\omega^2}, \quad b = \frac{\beta}{m\omega^2}, \quad c = \frac{\gamma}{m\omega^2}, \quad d = \frac{\delta}{m\omega^2}, \quad \text{and} \quad e = \frac{\epsilon}{m\omega^2}$$

and the prime is differentiation with respect to the non–dimensional time. The potential function can now be written in a pseudo–non–dimensional form as

$$V(x,y) = \frac{a}{2}x^2 + \frac{b}{4}x^4 + \frac{c}{2}y^2 + \frac{d}{4}y^4 + \frac{e}{2}x^2y^2 \tag{3.28}$$

Two separate forcing conditions will be considered

$$\frac{1}{m\omega^2}F(\tau) = \frac{1}{m\omega^2}G(\tau) = f\cos\tau \tag{3.29}$$

$$\frac{1}{m\omega^2}F(\tau) = f\cos\tau \quad \text{and} \quad \frac{1}{m\omega^2}G(\tau) = f\sin\tau \tag{3.30}$$

The system with the forcing in Equation 3.29 will be known as the synchronous system and the system with forcing in Equation 3.30 will be known as the nonsynchronous

system. The state–space representation of the synchronous system is

$$\begin{aligned}
\dot{x}_1 &= x_2 \\
\dot{x}_2 &= f\cos x_5 - zx_2 - ax_1 - bx_1^3 - ex_1x_3^2 \\
\dot{x}_3 &= x_4 \\
\dot{x}_4 &= f\cos x_5 - zx_4 - cx_3 - dx_3^3 - ex_1^2x_3 \\
\dot{x}_5 &= 1
\end{aligned} \tag{3.31}$$

For the nonsynchronous system, the equation for $\dot{x}_4$ will be changed to

$$\dot{x}_4 = f\sin x_5 - zx_4 - cx_3 - dx_3^3 - ex_1^2x_3 \tag{3.32}$$

Chapter 4

Computational Algorithm Implementation

Many of the techniques discussed in Chapter 2 are most effectively implemented in computer algorithms due to the multitude of calculations required. This chapter presents a detailed outline of the algorithms that were developed for each separate analysis method and how they interact with each other. Verification of these algorithms against both exact solutions and established research are given throughout the chapter. The computational implementation of the animations of the Poincaré mapping, phase portrait projections, and power spectra estimate techniques is discussed at the end of this chapter. This chapter also discusses some of the more basic elements of computer programming that were used in these programs such as software environments, file formats, and file reading and writing.

4.1 Software Environments

Matlab was chosen as the platform for implementing the numerical methods outlined in Chapter 2. However, structured programming in *Matlab* is prohibitively slow. Any program requiring repetition structures executes an order of magnitude or higher slower when programmed in *Matlab* compared to the C language. To avoid this, the C compiling abilities of *Matlab* were utilized through the use of mex-files. Mex-files provide the ability for C code to be compiled and run as a standard *Matlab* m-file. Many of the underlying routines in *Matlab* are written in this fashion. In a mex-file, C code can be written as

normal, but an additional function describing memory management and data structures is required as an intermediary. This function is commonly known as a mex function. A function call from the mex function calls the main program routine, and then the program executes as simply as if it was written only in C. While *Matlab* is slower at computation, the graphics capabilities are far superior than C. Any functions requiring graphing or pictorial representation were implemented in *Matlab*. Because of this, the data files were saved in *Matlab*'s mat-file format. Provisions for reading from and writing to this format are available from mex-files.

Since many of the functions written for the numerical methods outlined in Chapter 2 require similar subfunctions, i. e. data loading, varying control parameters, input/output, it is logical to use one main program so that common functions can be utilized accordingly. This "main" program was constructed such that each separate subroutine would be activated through the use of a "flag." The main program would execute the appropriate functions dependent on the value of the flag. This main program was written as a *Matlab* m-file. A "header" file is created each time the program is run for each flag type. These header file consists of program inputs, outputs, filenames, computation time, and other pertinent information.

The full program for the synchronous forcing can be found in Appendix C.2. The difference in the synchronous and nonsynchronous cases would be the alteration of the mex-file call for the Lie series and Lyapunov exponent function. Sample outputs of the header file for each flag type can be found in Appendix D starting on page 239.

A naming convention for all of the written data files was derived such that the file name consists of the flag type, system type, and then the system parameters and values. Under this system, the data loader function executes simple string manipulation to create the filename, locate the file, and then load the appropriate data. The header file also follows this naming convention with the exception that the file extension is .txt. All of the file naming was performed within a *Matlab* m-file. This could have been done from the C language, but number to string conversion is rather difficult to implement, and usually requires a priori knowledge of the number that is being converted. The decimal place is also lost upon conversion. The *Matlab* function 'num2str' is very easy to use and

retains decimal places. The 'strcat' function which performs string appending was also used extensively. Another advantage of using *Matlab*'s string manipulation is that the *Matlab* version of 'strcat' allows more than two strings to be appended in one function call, whereas the C version does not. Many of the C programs written therefore take any required file names as part of the input.

4.2 Lie Series Integration

As mentioned in Section 2.1, the computation of the infinitesimal generators is greatly aided by the use of a symbolic mathematics program. The *Maple* program provided in [3] was modified to generate the powers of U and place them in a matrix. The generated sequences were easily converted into equivalent C code using *Maple*'s 'codegen' command. The sequences generated for the higher powers of U for a fifth order system are long and computationally intensive. However, because the powers of U are computed by composition, there are many subexpressions that are repeated. The 'optimize' option in the 'codegen' command, which performs simple subexpression optimization, was utilized. The resulting C code was placed in a *Matlab* mex-file and compiled. The full *Maple* program can be found in Appendix A.

To ensure that the Lie series integration approach is valid, a *Matlab* solver comparison program was written. This program compares many of *Matlab*'s variable ODE solvers, Simulink fixed step solvers, a fourth–order Runge–Kutta scheme, and the Lie series approach. The Runge–Kutta and Lie approaches were coded in both *Matlab* and C. The computation time and an error estimate were calculated for each solver. The solver comparison program was written for a linear coupled mass vibration problem (Figure 4.1) and a first–order nonlinear differential equation with a known exact solution.

$$\dot{x} = 1 - x^2 \qquad x(0) = 0 \tag{4.1}$$

The exact solution to the above equation is $x(t) = \tanh t$. The parameters in the coupled mass problem were chosen so the system would be under–damped. The code for the

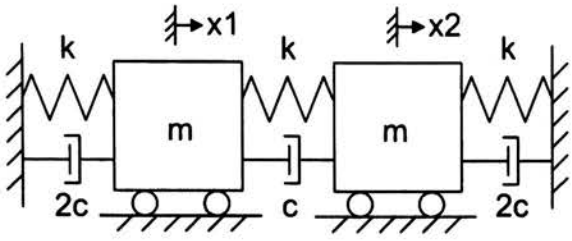


Figure 4.1: Coupled mass–spring–damper system

coupled mass program can be found in Appendix C.1. Similar code was written for the nonlinear differential equation. The results are shown in Table 4.1 and Table 4.2.

		Mass 1		Mass 2	
Solver	Time (sec)	Max. Error	Tot. Error	Max. Error	Tot. Error
ode45	0.638	1.806e-2	1.232e-1	2.017e-3	8.947e-3
ode23	2.217	1.838e-2	3.023e-1	2.146e-3	2.160e-2
ode113	1.066	1.793e-2	1.909e-1	1.915e-3	1.416e-2
ode5	0.178	6.474e-7	1.079e-5	8.257e-7	7.474e-6
ode4	0.137	6.963e-7	1.166e-5	7.872e-7	7.515e-6
ode3	0.121	2.482e-5	3.389e-4	1.119e-5	1.514e-4
RK4 m	2.238	2.665e-7	6.194e-6	1.293e-7	1.791e-6
RK4 mex	0.010	2.665e-7	6.194e-6	1.293e-7	1.791e-6
Lie2 m	0.857	2.745e-3	6.304e-2	9.395e-4	9.481e-3
Lie2 mex	0.028	2.744e-3	6.304e-2	9.395e-4	9.481e-3
Lie3 m	0.859	1.741e-4	2.441e-3	6.083e-5	5.796e-4
Lie3 mex	0.029	1.742e-4	2.443e-3	6.087e-5	5.801e-4
Lie4 m	0.863	3.174e-6	7.098e-5	1.062e-6	1.083e-5
Lie4 mex	0.028	3.230e-6	7.097e-5	1.051e-6	1.085e-5
Lie5 m	0.868	1.311e-7	1.832e-6	4.511e-8	4.375e-7
Lie5 mex	0.029	1.711e-9	3.803e-8	5.711e-10	5.829e-9
Lie6 m	0.871	1.711e-9	3.803e-8	5.711e-10	5.829e-9
Lie6 mex	0.029	1.711e-9	3.803e-8	5.711e-10	5.829e-9

Table 4.1: Solver results for the coupled mass–spring–damper system

The results shown in these tables indicate that all of the available solvers in *Matlab* are either less accurate, take more time, or both, compared to higher order Lie series approximations. It can also be seen that a Lie series iteration approach of fourth order or higher exceeds Runge–Kutta in accuracy. A sixth–order Lie series scheme was chosen for use in this study. Machine tolerances prevented a higher order due to the exponential nature of the series. The seventh–order step multiplication factor for a step size of 0.01

Solver	Time (sec)	Max. Error	Tot. Error
ode45	0.0609	3.770e-3	6.300e-3
ode23	0.0537	4.103e-3	1.106e-2
ode113	0.1962	4.737e-3	1.337e-2
ode5	0.1584	5.673e-14	6.458e-13
ode4	0.1269	1.477e-10	1.647e-9
ode3	0.1200	2.730e-8	2.977e-7
RK4 m	3.3438	1.477e-10	1.647e-9
RK4 mex	<0.0000	1.477e-10	1.647e-9
Lie2 m	0.9113	1.020e-5	8.226e-5
Lie2 mex	0.0300	1.020e-5	8.226e-5
Lie3 m	0.9169	6.935e-8	5.918e-7
Lie3 mex	0.0306	6.935e-8	5.918e-7
Lie4 m	0.9175	3.061e-10	2.115e-9
Lie4 mex	0.0313	3.061e-10	2.115e-9
Lie5 m	0.9241	2.484e-12	1.845e-11
Lie5 mex	0.0325	2.484e-12	1.845e-11
Lie6 m	0.9603	9.770e-15	1.129e-13
Lie6 mex	0.0341	9.770e-15	1.642e-13

Table 4.2: Solver results for the nonlinear differential equation

seconds is

$$\frac{0.01^7}{7!} = 1.974 \times 10^{-18} < \mathcal{E} \tag{4.2}$$

where $\mathcal{E}$ is the machine tolerance. The maximum machine tolerance on the computers used in this study was 2.2204×10^{-16}. This could have been circumvented using data types of more than one word length, but the increase in computation time to process these multi–word numbers would outweigh any gained accuracy.

4.3 Lyapunov Exponents

The *Maple* program presented in Section 4.2 was also modified to calculate the derivatives of the powers of U for use in the Jacobian matrix (dU in the program). Many of the subexpressions for U and dU are identical. However, no provisions for generating optimized C code for multiple arrays are available with *Maple*. The variable *all* was created which combines U and dU. However, some of the elements of *all* are not valid and some of the array indexing needs to be modified, so care must be taken when copying into the

mex-file. Computation time was decreased by 10–20% by using the optimized variables from the *all* array rather than those used separately by the U and dU arrays.

Lyapunov exponents should not be calculated until the transient portion of the solution has past. A switch was placed in the program so the user could specify how many seconds of the integrated simulation will be ignored before calculating the exponents. A provision was also added for calculating the absolute and relative convergence of the Lyapunov exponents.

A function was written for the Gram–Schmidt orthonormalization and placed within the same program. For numerical stability, the modified Gram–Schmidt was used [9]. In this method, only one inner product term is subtracted at a time. This function takes advantage of the Basic Linear Algebra Subroutines (BLAS) contained in LAPACK. LAPACK is a large, multiauthor Fortran library that *Matlab* uses to speed up matrix operations. Mex-files allow BLAS functions to be called directly as long as the file is compiled with the proper LAPACK library. Five BLAS functions were called from the Gram–Schmidt function; dgemm, dnrm2, dscal, ddot, and daxpy. The function dgemm is simply matrix multiplication. Upon implementation of this function, it was found that the arguments for the matrices must be switched in order for it to operate correctly. The dnrm2 function returns the Euclidean norm of a vector; dscal is scalar vector multiplication; ddot is the inner product function; and daxpy modifies a vector such that $\vec{y} = a\vec{x} + \vec{y}$ where a is a scalar. The Gram–Schmidt function also reset the B matrix from Section 2.4.

Since the Lie series iteration and Lyapunov exponent calculation rely on the same information, both were packaged together in a single mex-file. This mex-file also performs the mat-file writing for the time series data and the Lyapunov exponents, writing each of the data sets in two separate mat-files. The code for this program can be found in Appendix B.1. The 'sim' flag in the main program instructs the program to execute this C file, and gives 'syncmex.c' the proper input and output arguments. Inputs to the main program when using the 'sim' flag are the time span, time step size, initial conditions, the amount of time to ignore before calculating the Lyapunov exponents, and all of the equation parameters.

The Lyapunov exponent calculation was tested on a fourth–order non–autonomous linear equation (Equation 4.3) to test the algorithm's validity.

$$x^{(iv)} + 10x^{(iii)} + 35\ddot{x} + 50\dot{x} + 24x = cos(\omega t) \tag{4.3}$$

The general solution for the above equation is

$$x(t) = c_1 e^{-t} + c_2 e^{-2t} + c_3 e^{-3t} + c_4 e^{-4t} + c_5 \left(e^{\omega t} + e^{-\omega t}\right) \tag{4.4}$$

Thus the Lyapunov exponents are 0, -1, -2, -3, and -4. The convergence of the Lyapunov exponents for this system is shown in Figure 4.2. The function works correctly since the values of the Lyapunov exponents converge to their proper values.

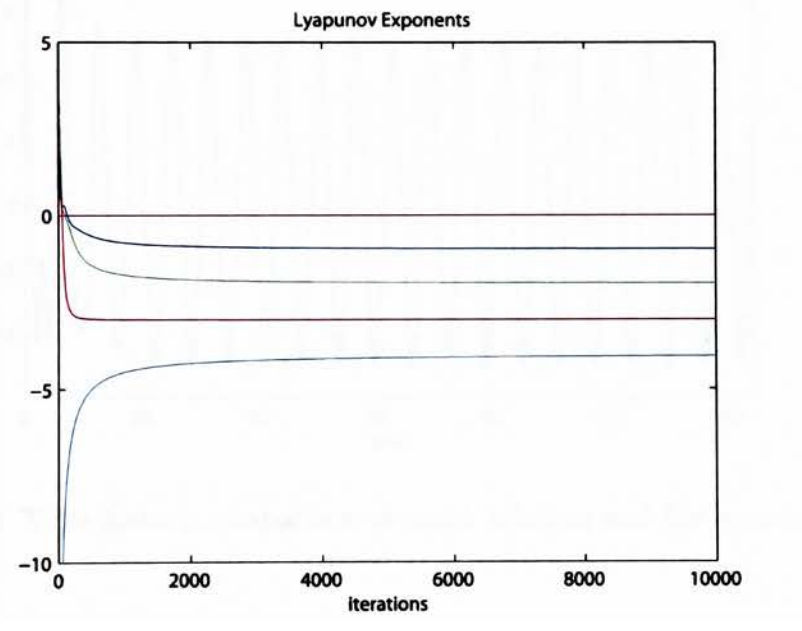


Figure 4.2: Convergence of the Lyapunov exponents for Equation 4.3

In the Lyapunov exponent calculations, it was assumed that values less than 10^{-6} were zero.

As a further check of the combined Lie series integration and Lyapunov exponent calculation, the parameters of Equation 3.27 were chosen such that the system reduced to two uncoupled simple harmonic oscillators. Exact analytical solutions are available for the equations when b, d, and e are equal to zero. The remaining parameters had values of $z = 0.3$, $a = 4$, $c = 6$, and $f = 1.5$. Figure 4.3 shown the exact analytical

solution compared to the implemented sixth–order Lie series approximation. The step size employed for this simulation was 0.1. Figure 4.4 shows the convergence of the absolute error of the Lie approximation as a function of time. It is clear that the Lie approximation converges past the transient portion of the solution. Figure 4.5 shows that the first four Lyapunov exponents converge to the linear eigenvalue of 0.15 as expected.

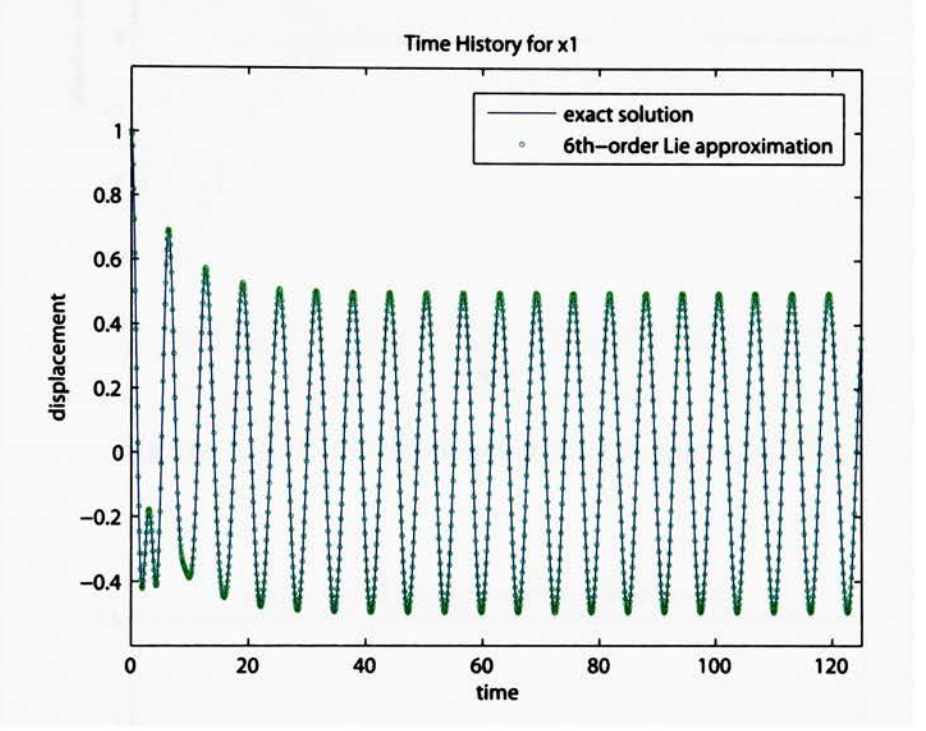


Figure 4.3: Time history comparison of exact solution and Lie approximation

4.4 Lyapunov Spectrum

The Lyapunov spectrum was created by loading the Lyapunov exponent data for each value of the control parameter within its range. The last values of each Lyapunov exponent are saved in a new data array. The current value of the control parameter is also placed in a new data array. A provision was added for determining the worst values of absolute and relative convergence. Since this function requires repetition based on the varying control parameter, it was also written in a mex-file. Upon implementation, it was found that loading the entire amount of data for each exponent was extremely, and

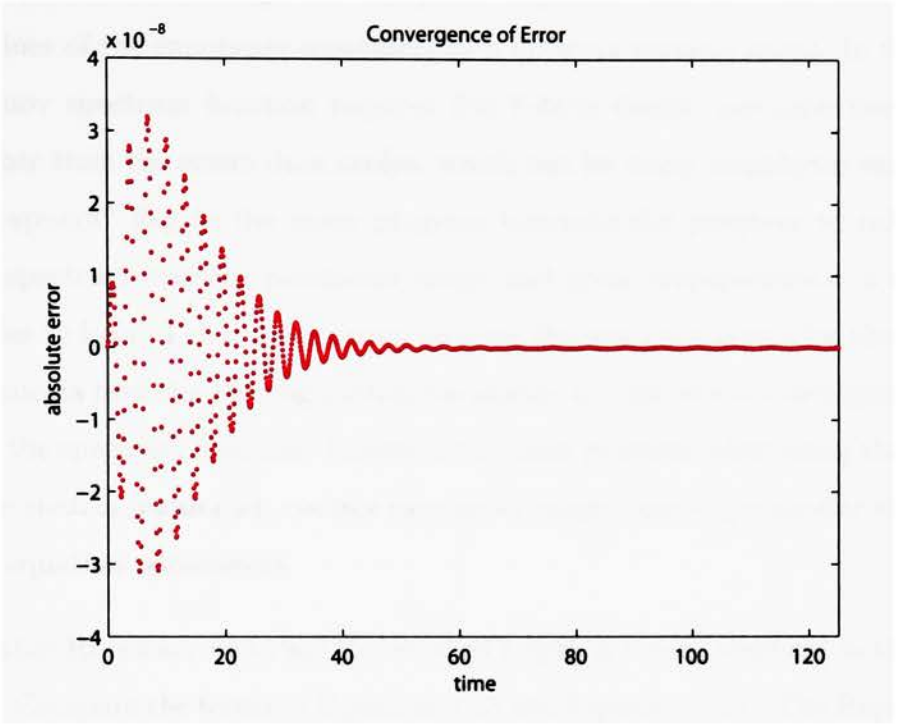


Figure 4.4: Error convergence for Lie approximation

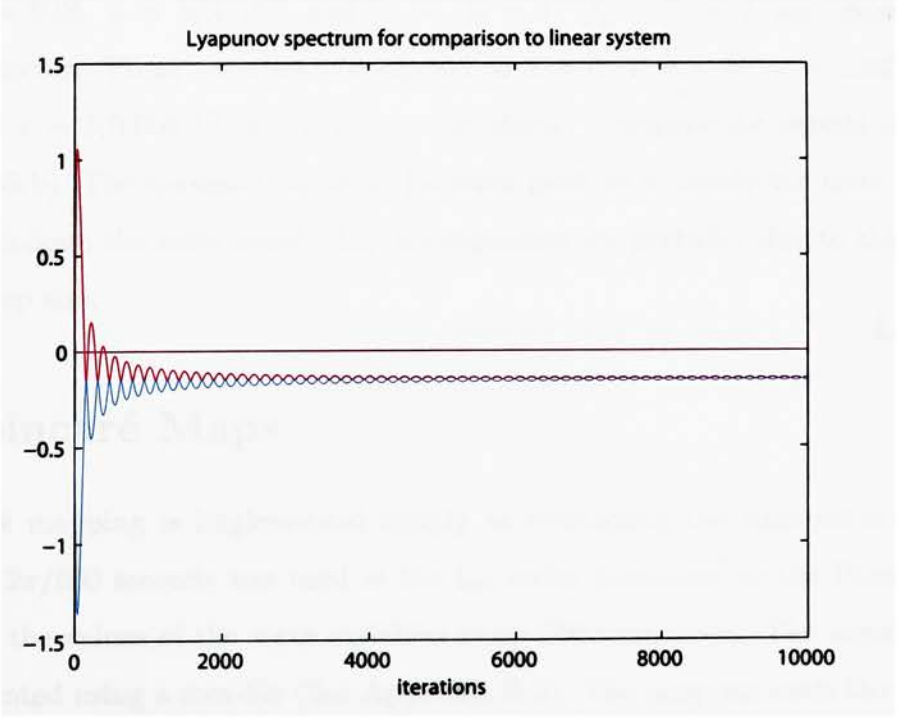


Figure 4.5: Convergence of Lyapunov exponents to proper linear eigenvalues

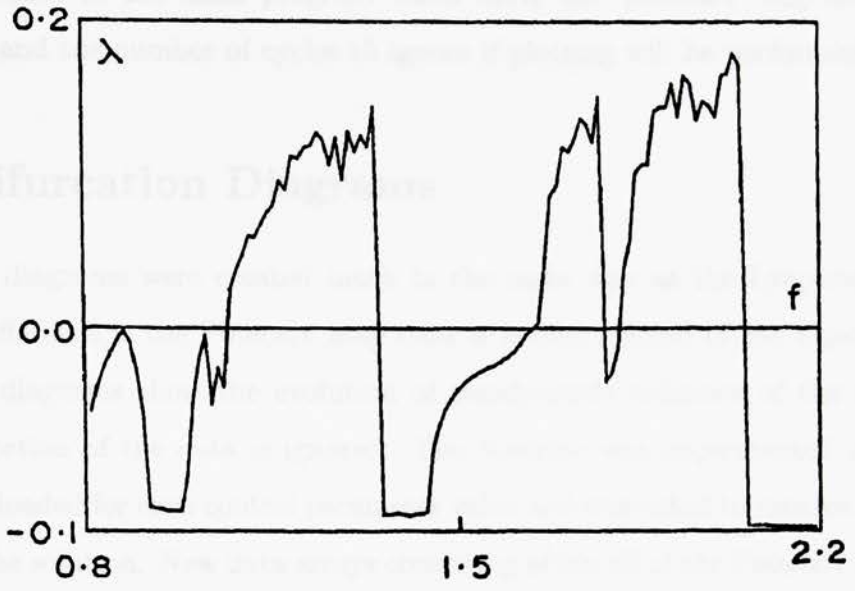
unnecessarily, time consuming. The Lyapunov exponent function was modified to save the last values of the exponents separately as a different variable name. In this method, the Lyapunov spectrum function requires five 8–byte double precision numbers to be loaded rather than the entire data arrays, which can be many megabytes each.

The 'lyapunov' flag in the main program instructs the program to calculated the Lyapunov spectrum over the parameter range, and gives 'lyapspecmex.c' a cell array of the data files to load. The main program receives the new data arrays for the final values of the exponents over the varying control parameter and the worst convergence values as outputs of the spectrum function. Inputs to the main program when using the 'lyapunov' flag are the control parameter, control parameter range, control parameter step, and the remaining equation parameters.

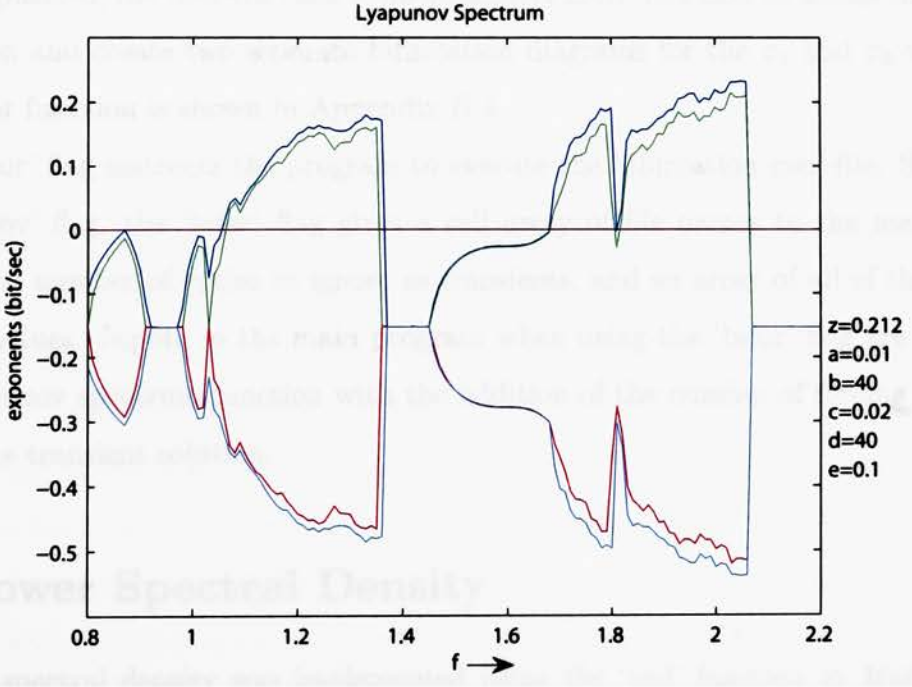
Recall that Rajasekar and Paul Raj studied a system nearly identical to the system in this study. Compare the forms of Equation 3.23 and Equation 3.27. The Rajasekar study [36] was repeated as a further validity check of the Lyapunov exponent and spectrum functions. The parameters used in their study were $A_1 = 0.005$, $A_2 = 0.01$, $\alpha_1 = 10$, $\alpha_2 = 10$, $\delta = 0.05$, $d = 3(A_1)^{1/2}$, and $\Omega_1 = \Omega_2 = 1$. $f_1 = f_2 = f$ was chosen as the control parameter. These variables correspond to $a = 0.01$, $b = 40$, $c = 0.02$, $d = 40$, $e = 0.1$, and $z = 3(0.005)^{1/2} \approx 0.21213$ in this study. Compare the results in Figures 4.6(a) and 4.6(b). The maximal exponent becomes positive in nearly the same locations and roughly follows the same trend. Any discrepancies are probably due to the differing parameter step size.

4.5 Poincaré Maps

The Poincaré mapping is implemented simply as resampling the time series data. A time step of $2\pi/500$ seconds was used in the Lie series iterations, so the Poincaré map is defined as the values of the state variables every 500 time steps. The algorithm was also implemented using a mex-file (See Appendix B.2). The program loads the 'sim' flag (time series) data based on the input file name. The 'poincare' flag in the main program instructs the program to execute this mex-file. If no output variables are specified, the



(a) Maximal Lyapunov exponent vs. control parameter f [36]



(b) Lyapunov spectrum for identical parameters as above

Figure 4.6: Lyapunov spectrum comparison

program plots the Poincaré map. An "ignored cycles" variable was created such that the plotter function ignores a certain number of points of the Poincaré map data before plotting. Inputs to the main program when using the 'poincare' flag are the system parameters and the number of cycles to ignore if plotting will be performed.

4.6 Bifurcation Diagrams

Bifurcation diagrams were created much in the same way as the Lyapunov spectrum. The only difference is the Poincaré map data is loaded instead of the exponents. Since bifurcation diagrams show the evolution of steady–state behavior of the solution, the transient portion of the data is ignored. The function was implemented in a mex-file. The data is loaded for each control parameter value and truncated to remove the transient portion of the solution. New data arrays consisting of the all of the Poincaré map data for each control parameter were then created from the loaded data. After these data arrays are fully populated, the mex-file calls a 'mexCallMATLAB' function to access the *Matlab* plot function and create two separate bifurcation diagrams for the x_1 and x_3 variables. The code for function is shown in Appendix B.4.

The 'bifur' flag instructs the program to execute the bifurcation mex-file. Similar to the 'lyapunov' flag, the 'bifur' flag gives a cell array of file names to the mex-file. It also gives the number of cycles to ignore as transients, and an array of all of the control parameter values. Inputs to the main program when using the 'bifur' flag are identical to the Lyapunov spectrum function with the addition of the number of forcing cycles to ignore as the transient solution.

4.7 Power Spectral Density

The power spectral density was implemented using the 'psd' function in *Matlab*. The 'psd' function uses Welch's modified periodogram estimation discussed in Section 2.5. The inputs for the 'psd' function are FFT size, window size and type, sampling rate, and the amount of overlap. The 'mean' detrending flag was also used to remove any offset in

each data window before calculating the periodogram. A Hamming window was used as the windowing function. The goal of using the PSD is to show the large magnitude peaks due to periodic and quasiperiodic signals. Thus, a 10 dB increase in the relative sidelobe level was deemed more important than the marginal decrease in transition bandwidth compared to the use of a Hann window.

The 'freq' flag in the main program instructs the program to execute the appropriate function for calculating the power spectra, and loads the time series data and truncates according to the value of FFT size. The implementation was such that the amount of periodogram overlap is one quarter of the window size. The size of the data is such that eight separate periodograms could be calculated with the overlap.

$$\text{data size} = 8 \times \text{window size} - 7 \times \text{overlap} \tag{4.5}$$

The data is taken from the end of time series data. Care must be taken so that the FFT size is small enough to prevent the amount of data from including some or all of the transient portion of the solution. The resampling was implemented through *Simulink*'s discrete solver. The solver linearly interpolates between the input data points and outputs the interpolated values. Two separate power spectra are created corresponding to the x_1 and x_3 variables. Similar to the 'poincare' flag, data is plotted if no output variables are specified. Inputs to the main program for the 'freq' flag are the system parameters, FFT size, window size, and resampling rate if necessary.

The FFT and window size used in this study was 65536 which along with a step size of $2\pi/500$ gave a frequency resolution of 0.00726 rad/sec. Frequency components as high as 250 rad/sec can be detected. The primary concern with PSD calculation is detecting the subharmonic frequencies that result from period–doubling. Since the forcing frequency is effectively 1 rad/sec, most of the data frequency data above 1 rad/sec is superfluous. Unnecessary computation time could be removed from the FFT calculation by using a smaller FFT size, or the frequency resolution could be improved if the data was resampled with a larger time step.

4.8 Alternate Graphical Representations

4.8.1 Poincaré Map and PSD Animation

Examining the Poincaré maps and PSD graphs can be extremely time consuming if the control parameter is varied over a large range with a small increment, since a map and the PSD are calculated at each value. Bifurcation diagrams alleviate some of the tediousness associated with the Poincaré maps, but it can be difficult, if not impossible, to distinguish chaotic behavior from quasiperiodicity with the examination of only one state variable. Fortunately, this can be circumvented by using the animation capabilities within *Matlab*. Each frame in the animation can be a different map or PSD for a certain control parameter. Thus, the frame in the movie can step though time as the analysis results are stepping through the control parameter. For the Poincaré maps, this can almost be considered as a two–dimensional bifurcation diagram, in that the results of *two* of the state–variables can be seen versus the varying control parameter. For the PSD, it is quite easy to detect a period–doubling bifurcation since new peaks appear in PSD movie frame.

4.8.2 Phase Plane Projection Animation

Phase plane projections can also be useful as a visual tool. While the full phase portrait for the integrated trajectories cannot be visualized due to the inherent four–dimensional nature, two phase plane projections can show the steady–state shapes of each orbital path. Each one of the phase portraits corresponds to one of the equations shown by Equation 3.27. Examining all of these phase portraits is limited in the same manner as the Poincaré maps and the PSD graphs. Thus, each phase portrait was animated in a similar manner. Period–doubling can be detected on in the phase portrait movies easily. A closed orbit will suddenly split into two similarly shaped orbits. The new orbit will still be closed signifying that it is indeed a periodic solution.

4.8.3 Animation Implementation

The functions for the phase portrait movie and the Poincaré movie were partially implemented in mex-files. The major difference between these movie functions and the frequency movie function is the axis and plot scaling control. Scaling was performed to ensure that the movies play smoothly and don't "jump around" on different plot scales. These functions load all the data for the whole range of the parameters to determine the maximum plot size so that the individual plots are all viewable and don't "fall" off the screen.

In both of these functions, the entire data set is not of interest. For the phase portrait movie, the data is truncated such that the phase portrait will only consist of the last 8001 data points or 16 drive cycles, since the full phase portrait can be lengthy if a large number of drive cycles are integrated. The 8001 point value was chosen such that a period-16 solution can be viewed, but a chaotic phase portrait would not be filled in excessively. Conceivably a chaotic phase portrait would look nearly solid if enough drive cycles were shown. For the Poincaré movie, the transient points of the maps should be removed.

Because the full data in each case needs to be initially loaded and truncated to determine the movie sizes, it is not efficient to reload all of the data again and truncate before capturing each frame of the movies. Speed will be gained two–fold by writing temporary data files consisting of the truncated data sets. Loading time will be reduced by loading a much smaller file, and the additional computation time associated with re–truncating the full data set will not be necessary. The code for the phase movie mex-file and the Poincaré movie mex-file can be found in Appendices B.5 and B.6.

The flags to activate all these functions are 'phasemov', 'freqmov', and 'poinmov'. The 'freqmov' and 'poinmov' flags essentially work the same as the 'freq' and 'bifur' flags. The 'phasemov' and 'poinmov' flags call their respective mex-file components inputting the names of the data files and the temporary file names as separate cell arrays. The 'poinmov' flag also indicates to the mex-file the amount of cycles to ignore as transients. Each of these program flags load the appropriate data into a plot window capturing the

individual movie frame. The value of the varying control parameter is printed in an upper corner of the frames for all of the movies. Inputs to the main program when using all of these flags are the control parameter, control parameter range, control parameter step, and the remaining equation parameters. For the 'poinmov' flag, a variable containing the number of cycles to ignore as transients is required as a separate input.

Chapter 5

Synchronous System Analysis Results

5.1 Numerical Modeling

All numerical integrations were performed with a time step size of $2\pi/500$ or approximately 0.0126 seconds. The equations were integrated for 3000 forcing cycles, the first 500 of which were considered part of the transient solution for both the Lyapunov exponents and the Poincaré maps. In some cases, simulations resulting in interesting chaotic behavior were performed for a longer amount of time in order for the Poincaré maps to be better defined. The longer simulations will be noted in their respective figures.

Over 66,000 different simulations were performed throughout the course of this study. The average execution time for a simulation, including the system integration, Lyapunov exponent calculation, Poincaré map definition, and power spectral density estimate, was approximately 22 seconds on a system with a processor clock speed of 2.8 GHz and 2 MB of system cache.

Approximately 100 MB of data was generated during this time. Thus the 66,000 simulations represent a total processing time of over 400 hours, during which over 6600 GB of data was generated. The time and data estimates exclude the time required and data generated for the creation of the bifurcation diagrams, Lyapunov spectrums, and the total number of animations. The computer codes shown in the appendices represent many revisions in which the codes were subsequently rewritten more efficiently in order to achieve such a relatively low computation time.

5.1.1 Potential Function

The potential function shown Equation 3.28 can be considered symmetric in that a variable change of $x = y$ and $y = x$ results in the same equation up to the arbitrary constants. Because of this, not all of the variables in the potential function equation are required as control parameters. Scanning through the parameter c with a certain values of the other parameters can be equivalent to using a as the control parameter with corresponding values for the other parameters. Thus, the system dynamics can be investigated by only using a, b, and e as the control parameters.

It was also necessary to ensure that b, d, and e remained positive throughout the course of this study. While negative values of b, d, and e can be interesting for certain parameter combinations, overall, they result in time history solutions that are unbounded. Rajasekar and Paul Raj avoided the unbounded solutions in the third studied case by limiting the value of the forcing amplitude [36]. This ensured that their solutions did not approach the unstable equilibria points existing due to the negative values of b and d. Since this study focuses on varying the potential field constants rather than the forcing amplitude, it could not be ensured that the solutions would remain bounded for all of the different potential fields if b, d, and e were allowed to become negative.

5.1.2 Parameter Scans

In most cases, parameters were scanned with both positive and negative steps to identify any hysteresis loops present within the system. The initial conditions for the beginning of the parameter scan were all zeros if the parameter value was zero, 0.01, or 0.001. Otherwise, the final system state in the previous parameter value simulation was used as the initial conditions for the newly incremented or decremented parameter simulation. Most of the simulations performed used a parameter variation size of 0.01. In some cases, a finer size of 0.001 was used in order to better capture rapid parameter–based variations in the dynamics. In all of the simulations, z and f, the non–dimensional damping and forcing amplitude terms in Equation 3.27, were held fixed at 0.3 and 1.5 respectively.

A list of the parameter variations for the synchronous system is given in Table 5.1.

The choice of the parameter combinations was driven by qualitative changes in the shape of the potential field. In this sense, the parameter combinations are arbitrary up to the point of qualitatively affecting the potential field.

5.2 Scans of Parameter a

The parameter a in the potential equation controls the creation of two of the potential wells that can exist in the system. When a is negative with b positive, two symmetric wells exist on either side of the y-axis much like the wells shown in Figure 3.1.

5.2.1 Symmetric Cubic Terms

The first several scans through the a parameter were performed with the remaining system parameters having values of $b = 10$, $c = 0.1$, $d = 10$, and $e = 0.25$. The cubic uncoupled terms in this form are symmetric. The potential wells created by an increasingly negative value of a are fairly weak in the sense that the larger value of the quadric term overtakes the negative nature of the well very rapidly. Thus the wells are not very "deep".

The bifurcation diagrams when a is varied negatively from 0 to -10 are shown in Figures 5.1 and 5.2. The resulting Lyapunov spectrum is shown in Figure 5.3. The bifurcation diagrams show a very rapid period–doubling when $a = -4.02$. The period–doubling culminates in chaotic behavior with a maximal Lyapunov exponent of approximately 0.065, which is relatively weak chaos. The Poincaré maps for $a = -5.23$ in the chaotic region are shown in Figure 5.4. This chaotic region has very strong period–2 harmonic content, signifying probable merging of two chaotic attractors. However this merging is interrupted before the attractor becomes fully developed by a stable period–2 orbit that halves to a period–1 solution. The period–1 solution is persistent over the parameter space from -5.63 to -7.9. At $a = -7.91$, the period–1 solution loses stability and a period–2 solution results from a period–doubling bifurcation. The period–doubling is interrupted by a significant "jump" back to a period–1 solution. This jump is associated

Number	Parameter	Start	Stop	Step	a	b	c	d	e
1	a	0	-10	-0.01	-	10	0.1	10	0.25
2	a	-10	0	0.01	-	10	0.1	10	0.25
3	a	0	10	0.05	-	10	0.1	10	0.25
4	a	0	1.5	0.01	-	10	0.1	10	0.25
5	a	1.5	-8	-0.01	-	10	0.1	10	0.25
6	a	0	-12	-0.01	-	5	0.1	10	0.25
7	a	-12	0	-0.01	-	5	0.1	10	0.25
8	a	0	10	0.01	-	5	0.1	10	0.25
9	a	0	-12	-0.01	-	10	0.1	10	0.5
10	a	-12	0	0.01	-	10	0.1	10	0.5
11	a	0	-10	-0.01	-	0.1	-2	0.5	0.25
12	a	-0.8	0	0.01	-	0.1	-2	0.5	0.25
13	a	0	10	0.01	-	0.1	-2	0.5	0.25
14	b	0.01	35	0.01	-1	-	0.1	10	0.25
15	b	0.001	1	0.001	-1	-	0.1	10	0.25
16	b	10	2.2	-0.01	-1	-	0.1	10	0.25
17	b	0.01	16	0.01	-1	-	-5	0.5	0.25
18	b	0.01	12	0.01	-4.5	-	-3.5	0.25	1.25
19	b	0.01	20	0.01	-4.5	-	-2.25	0.25	1.25
20	b	0	1	0.001	2	-	3	0.25	0.1
21	b	0	1	0.01	2	-	3	0.5	0.75
22	b	0	1	0.01	2	-	4	1	0.75
23	b	0	1	0.01	2	-	4	0.5	0.75
24	b	0	1	0.01	2	-	4	0.5	1
25	b	0	1	0.01	2	-	4	0.75	0.75
26	e	0	40	0.01	-5.5	10	0.1	10	-
27	e	0	20	0.01	-1.5	0.1	-2	0.5	-
28	e	0	0.9	0.001	-1.5	0.1	-2	0.5	-
29	e	0	0.2	0.001	2	0.05	3	0.125	-
30	e	0	0.5	0.001	2	0.1	3	0.25	-
31	e	0	1	0.01	2	0.75	3	0.5	-
32	e	0	1	0.01	2	0.75	4	0.5	-
33	e	0	1	0.01	2	0.75	4	0.75	-
34	e	0	1	0.01	2	0.75	4	0.5	-
35	e	0	1	0.01	2	0.75	4	1	-

Table 5.1: Parameter variations for the synchronous system

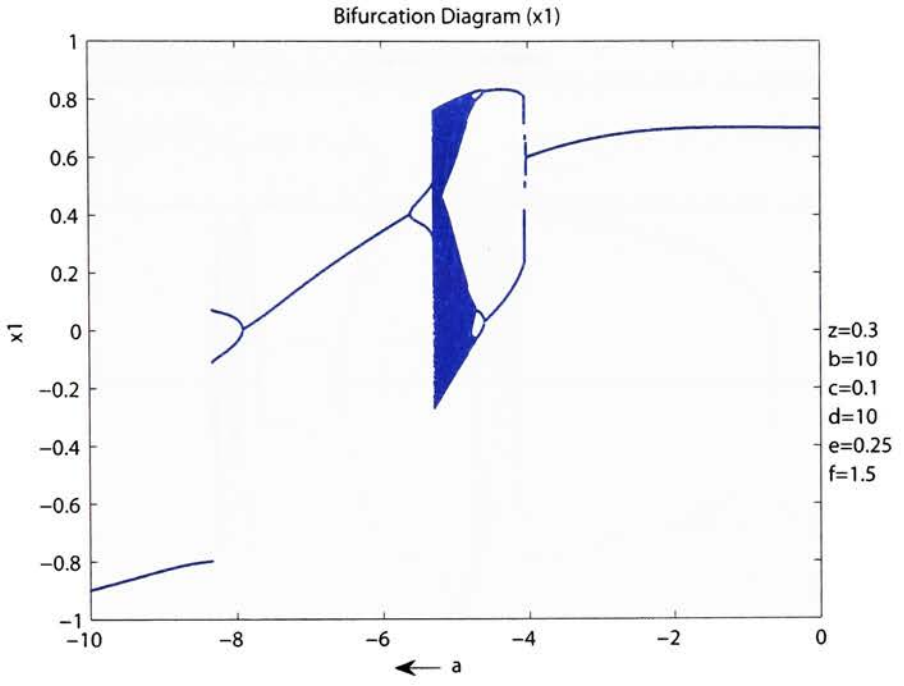


Figure 5.1: Bifurcation diagram for negative a scan with symmetric cubic terms (x_1)

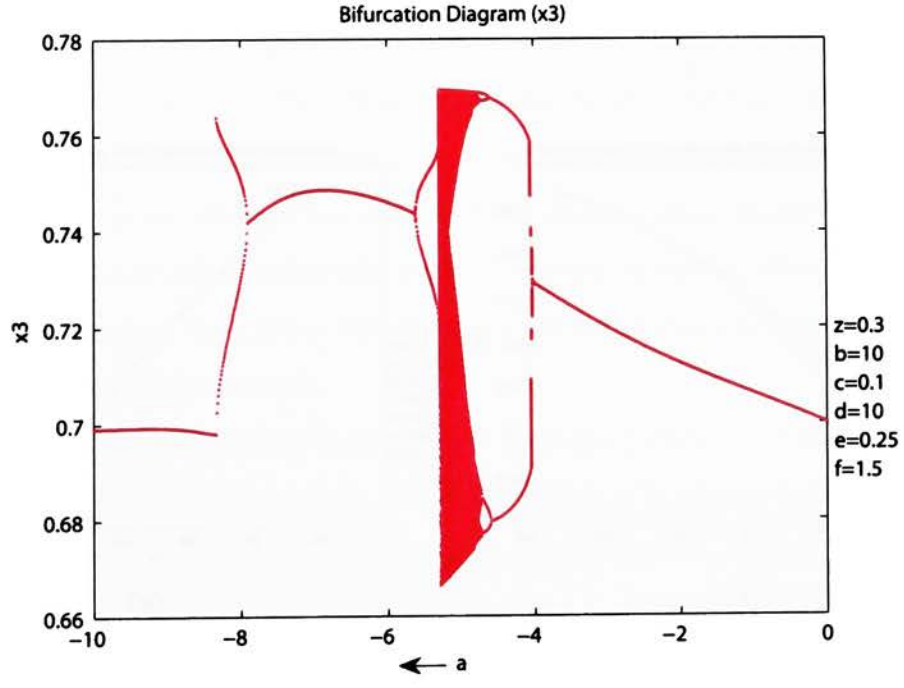


Figure 5.2: Bifurcation diagram for negative a scan with symmetric cubic terms (x_3)

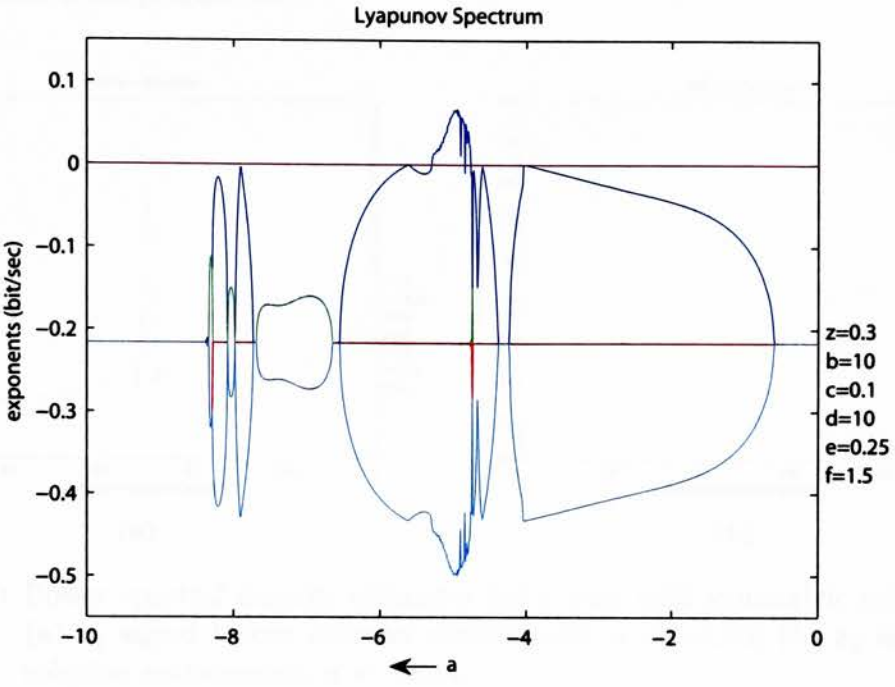
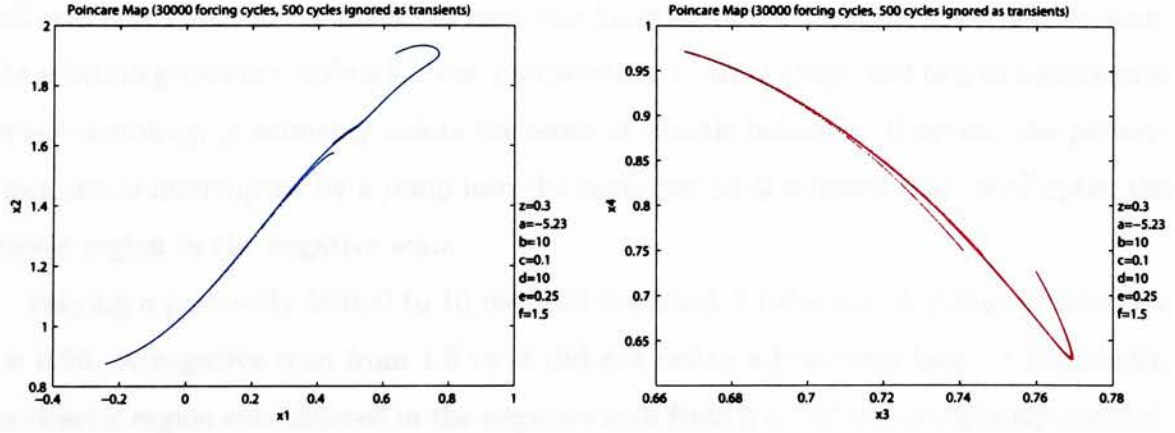


Figure 5.3: Lyapunov spectrum for negative a scan with symmetric cubic terms



(a) (b)

Figure 5.4: Chaotic Poincaré map projections for $a = -5.23$ in scan with symmetric cubic terms corresponding to: (a) x_1–x_2 plane; (b) x_3–x_4 plane.

with a 20 dB drop in the signal power, a result of the solution becoming entrained in one of the potential wells (Figure 5.5).

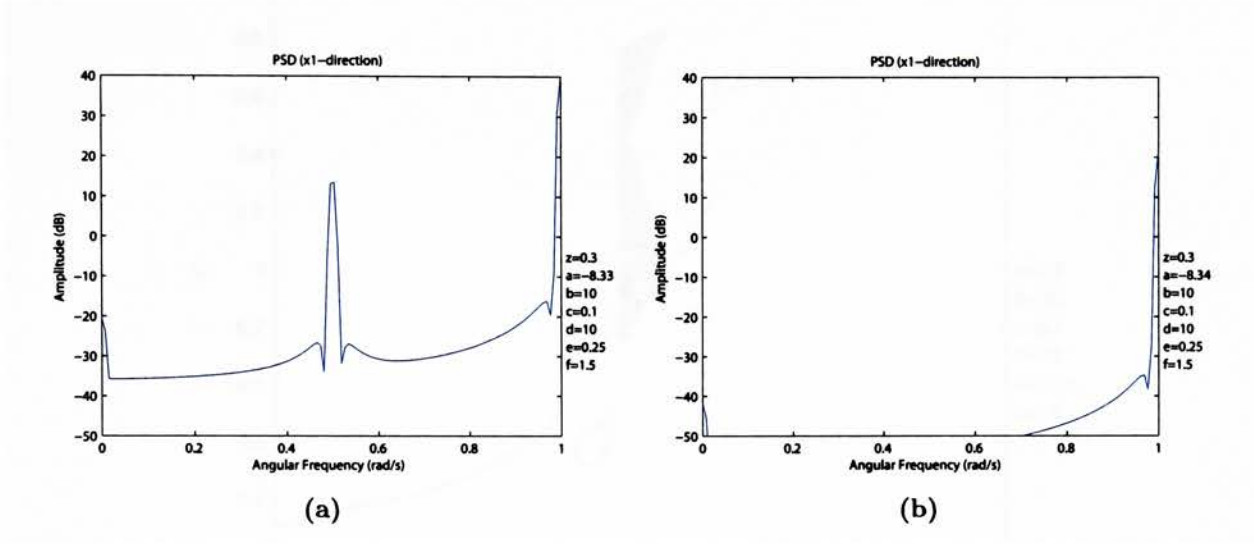


Figure 5.5: Power spectral density estimates for a scan with symmetric cubic terms: (a) x_1 signal before solution entrainment, $a = -8.33$; (b) x_1 signal after solution entrainment, $a = -8.34$.

The positive scan from -10 to 0 has bifurcation diagrams and a Lyapunov spectrum that are identical to the negative scan when the value of a is greater than approximately -5.4. Compare the shapes of Figures 5.6–5.8 to 5.1–5.3. The same region of chaos exists, and the exit from this region is through a period–halving phenomena. The low amplitude well–entrained solution is stable beyond the location of the jump in the negative scan. The solution grows in amplitude, then jumps into a different shape and begins a successive period–doubling, presumably before the onset of chaotic behavior. However, the period–4 solution is interrupted by a jump into the same period–2 solution that interrupted the chaotic region in the negative scan.

Varying a positively from 0 to 10 resulted in period–1 behavior. A jump occurred for $a = 0.96$. A negative scan from 1.5 to -8 did not define a hysteresis loop. Additionally, the chaotic region encountered in the negative scan from 0 to -10 was completely avoided. This indicates that the basin of attraction for the chaotic attractor may be rather small. A period–doubling and halving was detected in this negative scan. Otherwise the solution was completely period–1. All of the jumps in this particular variant of the system were accompanied by near positive peaks in the Lyapunov spectrum.

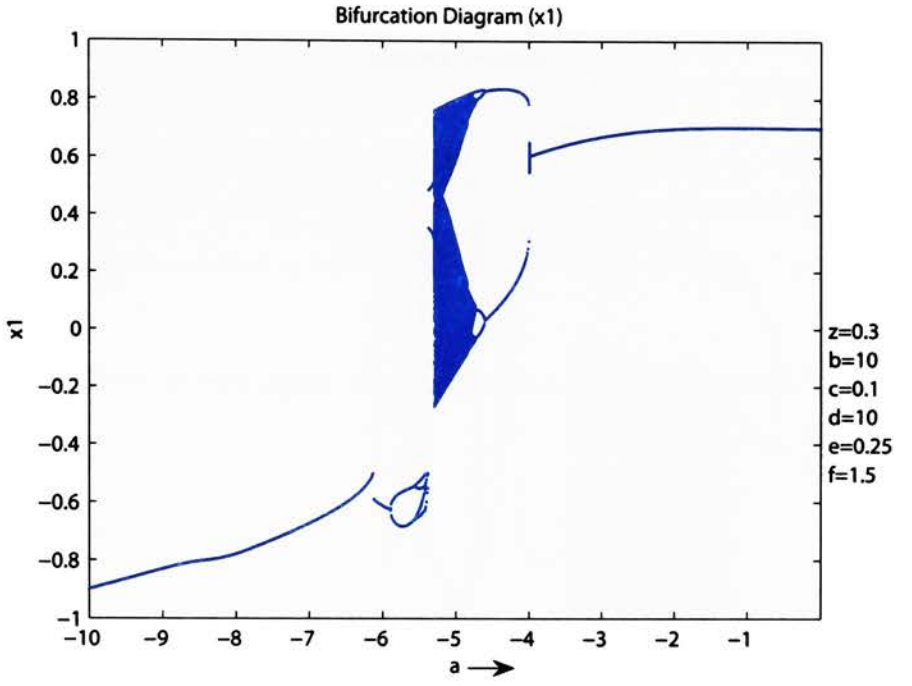


Figure 5.6: Bifurcation diagram for positive a scan with symmetric cubic terms (x_1)

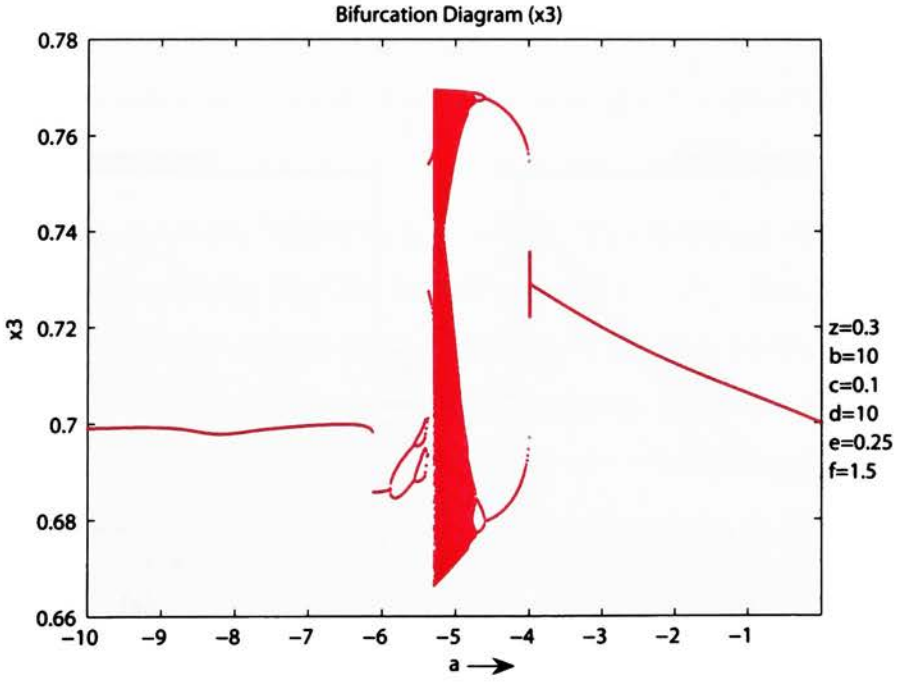


Figure 5.7: Bifurcation diagram for positive a scan with symmetric cubic terms (x_3)

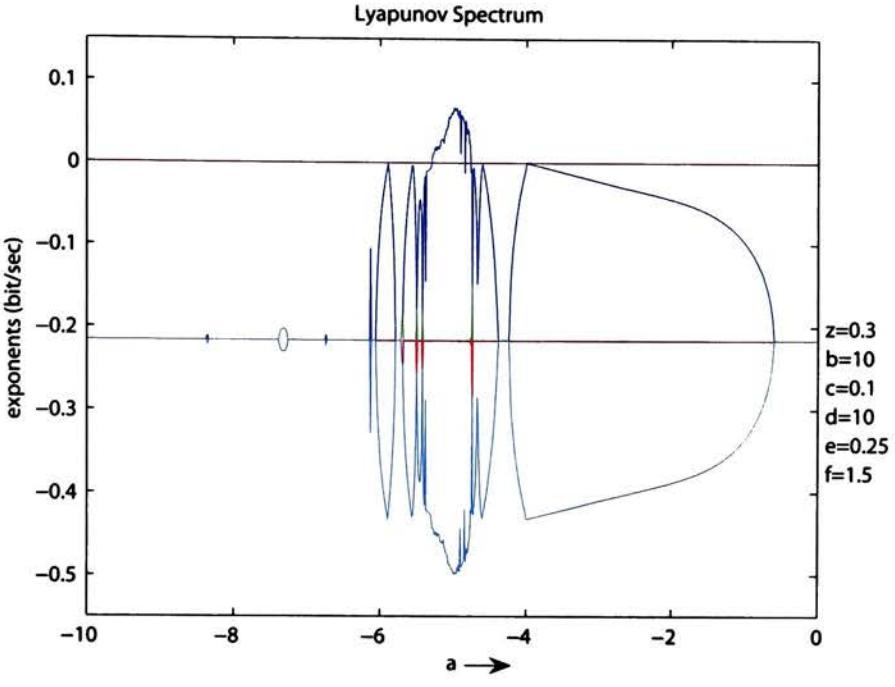


Figure 5.8: Lyapunov spectrum for positive a scan with symmetric cubic terms

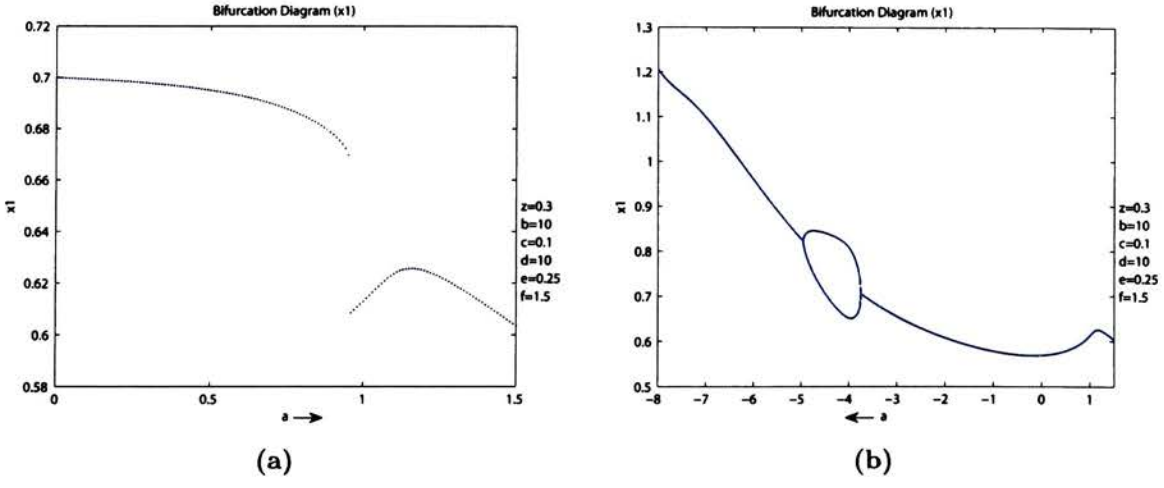


Figure 5.9: (a) Positive scan showing jump behavior; (b) Negative scan showing avoidance of chaotic behavior.

5.2.2 Nonsymmetric Cubic Terms

For the second group of a scans, the constants were reformulated such that b and d are no longer equal. The value of b was changed to 5 while all of the other variables retained the same values as the previous scan. The system starts in a period–1 solution.

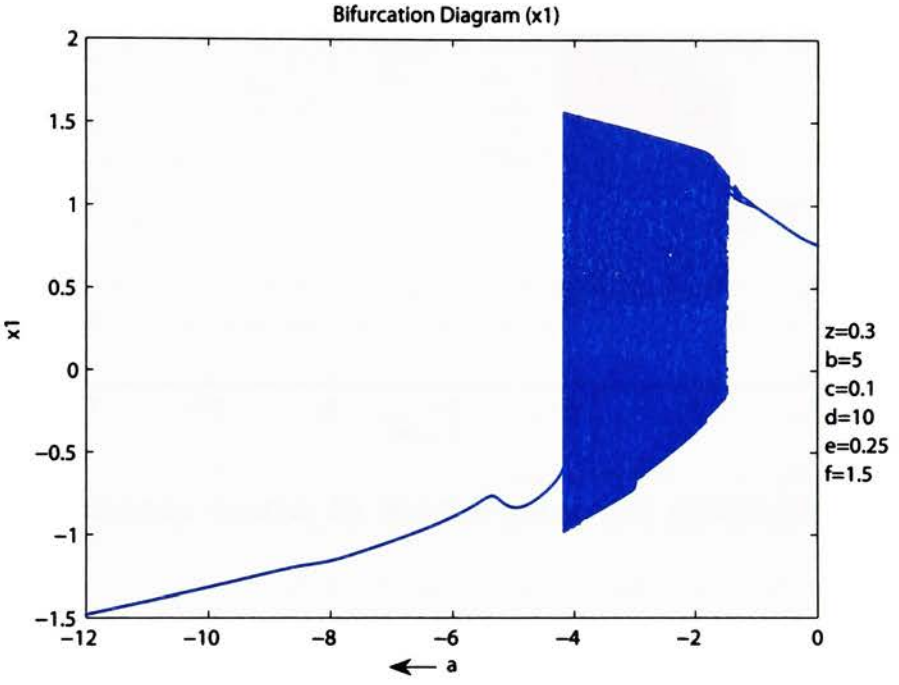


Figure 5.10: Bifurcation diagram for negative a scan with nonsymmetric cubic terms (x_1)

A period–doubling succession begins at $a = -1.05$. The doubling reaches a period–4 solution before interrupted by very low intensity chaos at -1.35. This brief chaos then causes the solution to revert to period–2. The period–2 solution enters a chaotic region through a crisis event. The maximum Lyapunov exponent associated with this chaotic region is approximately 0.2. The Poincaré maps for $a = -3.8$ in the chaotic region are shown in Figure 5.13.

Very brief periodic windows exist within this broadband region of chaos. Towards the end of this chaotic region, the solution begins to oscillate between two different chaotic attractors. The oscillations persist relatively briefly in the parameter space before the system becomes period–1 due to crisis. The exit from chaos is also characterized by the

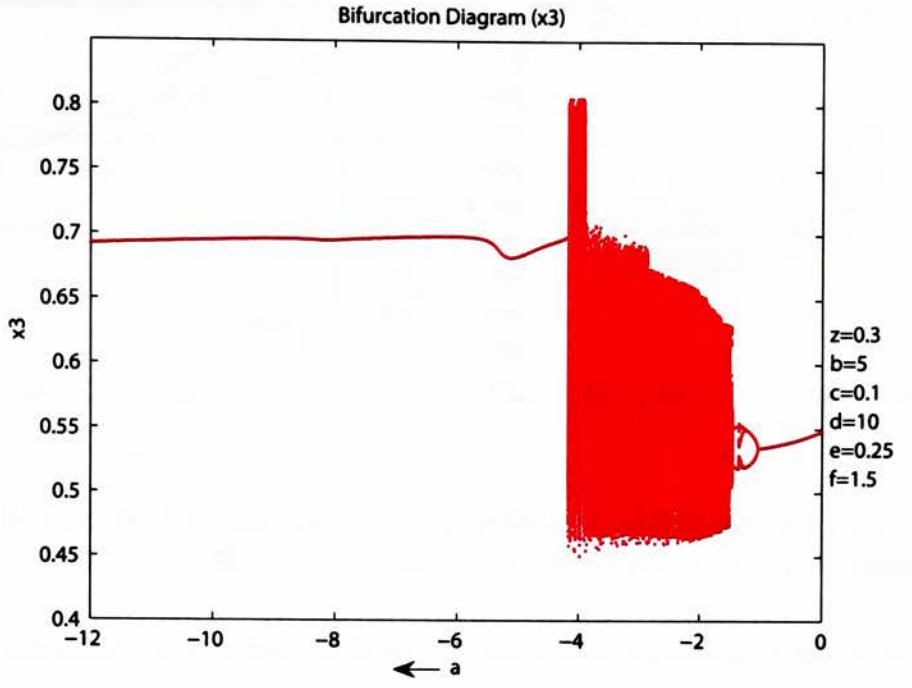


Figure 5.11: Bifurcation diagram for negative a scan with nonsymmetric cubic terms (x_3)

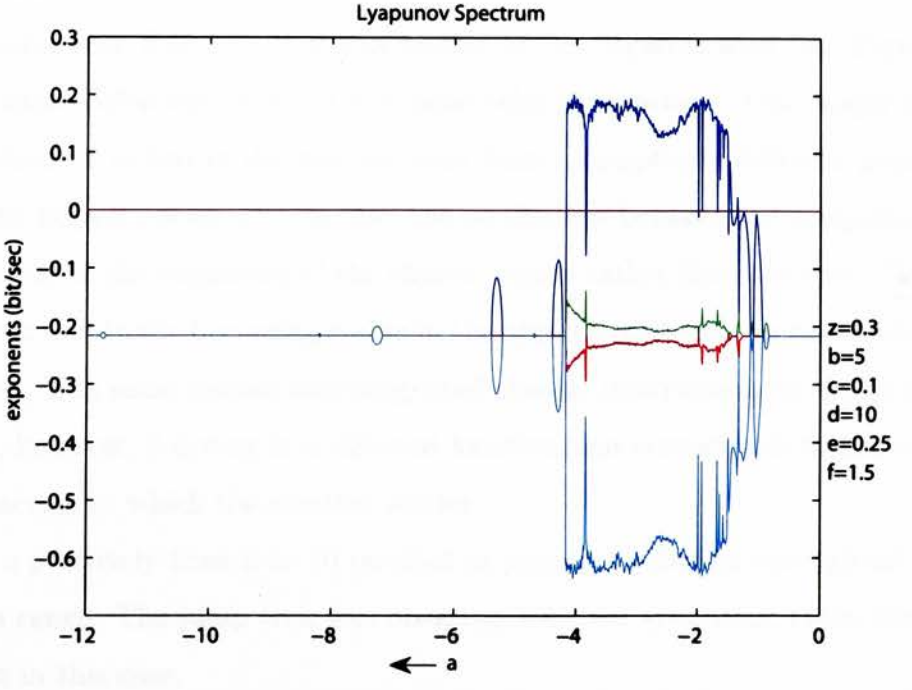
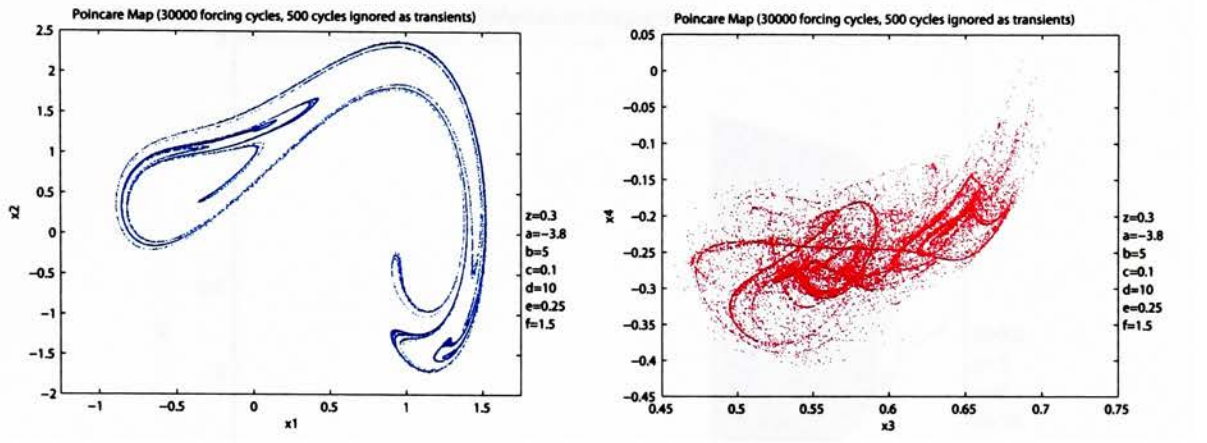


Figure 5.12: Lyapunov spectrum for negative a scan with nonsymmetric cubic terms



(a) (b)

Figure 5.13: Chaotic Poincaré map projections for $a = -3.8$ in scan with nonsymmetric cubic terms that correspond to: (a) x_1–x_2 plane; (b) x_3–x_4 plane.

same small amplitude low power cycle as the symmetric case, which again is probably the result of well entrainment. This reduction in power differs from the symmetric case in that the solution decays gradually rather than the extreme 20 dB jump previously observed. None of the jumps observed in the symmetric case were seen with the current parameter values.

The positive scan from -12 to 0 was similar to the negative scan (see Figures 5.14–5.16). The same behaviors occur in the same relative locations. One major difference is that the chaos is exited in the positive scan from a completely different point than it entered in the negative scan. In this case, the oscillations between the competing chaotic attractors occur in the beginning of the chaotic region rather than the end. This results in the solution eventually becoming stable in the opposite attractor as the solution in the negative scan. The same period–doubling/brief chaotic interruption occurs in the range of -1.5 to -1; however, it occurs in a different location that corresponds to the competing chaotic attractors in which the solution settles.

Varying a positively from 0 to 10 resulted in period–1 behavior through all values of a within the range. The jump that was observed with the symmetric cubic components did not exist in this case.

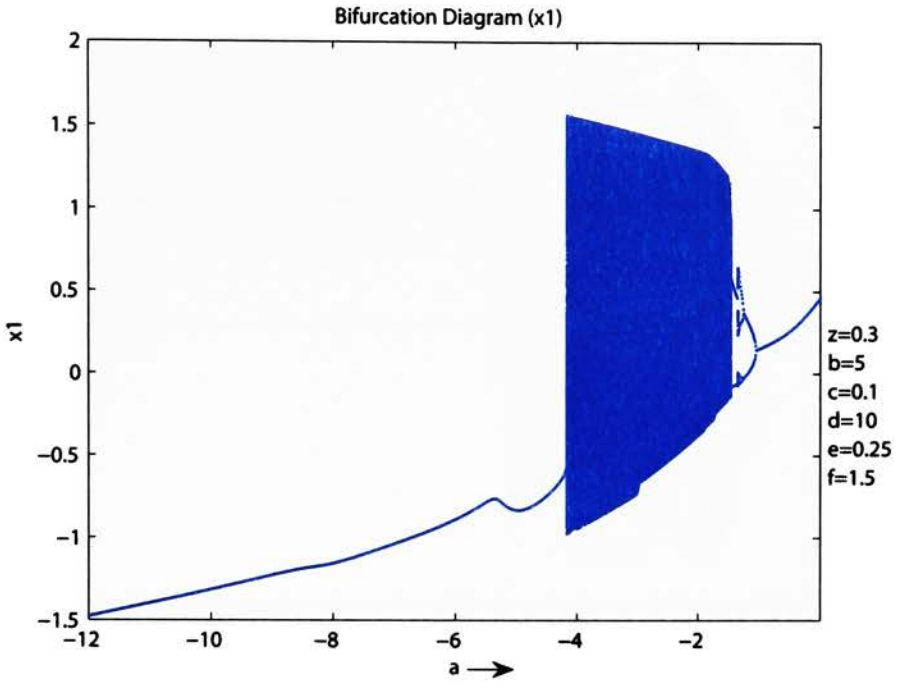


Figure 5.14: Bifurcation diagram for positive a scan with nonsymmetric cubic terms (x_1)

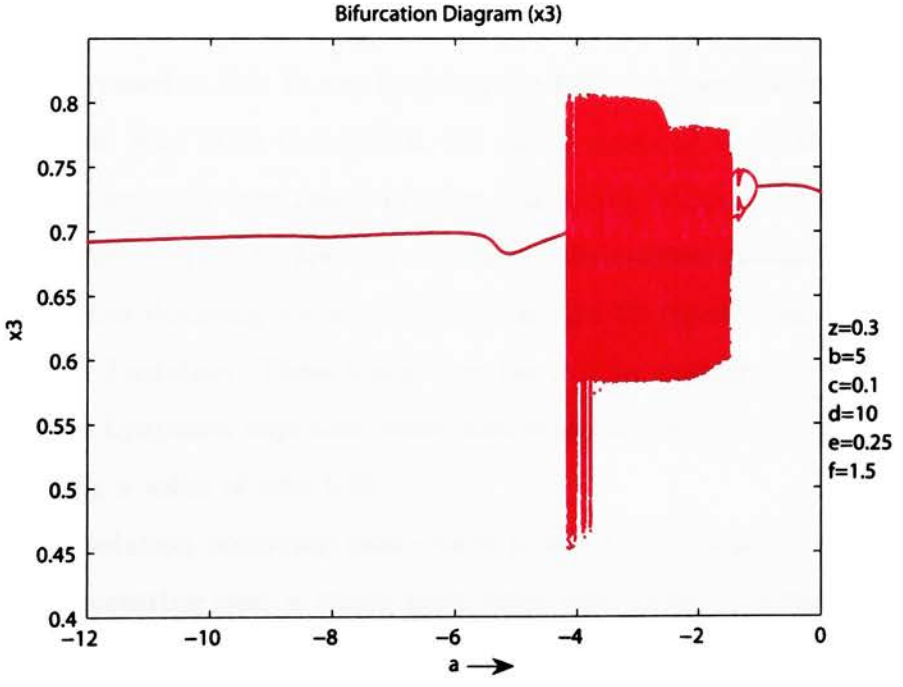


Figure 5.15: Bifurcation diagram for positive a scan with nonsymmetric cubic terms (x_3)

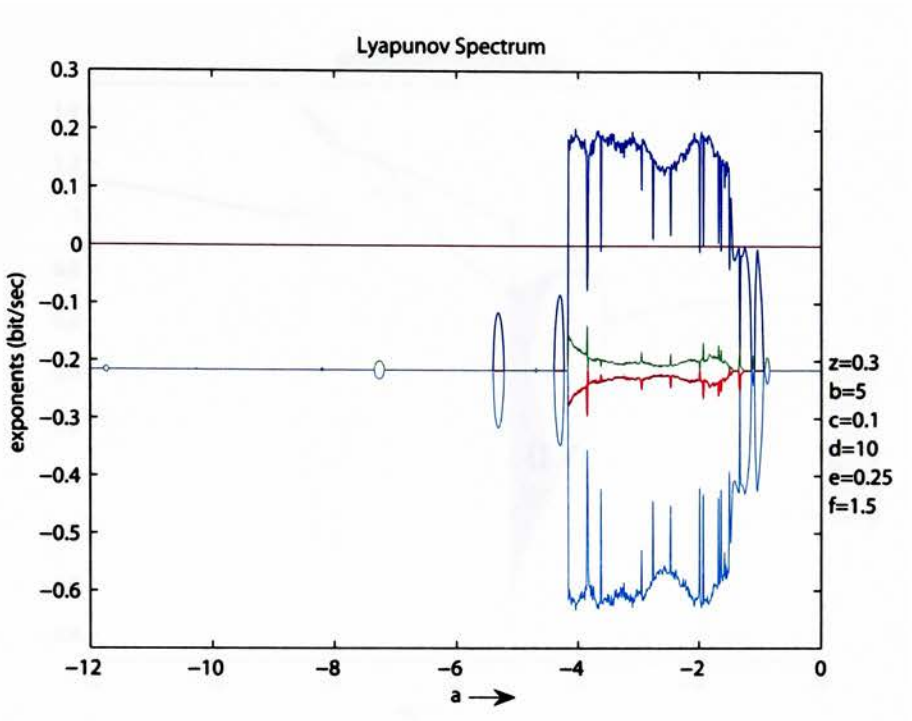


Figure 5.16: Lyapunov spectrum for positive a scan with nonsymmetric cubic terms

5.2.3 Symmetric Cubic Terms, Increased Coupling Strength

For this a scan, the cubic terms are again symmetric with $b = d = 10$, but the coupling parameter e is increased to 0.5. It was found in the first symmetric scan that the x_3–x_4 projections changed very little throughout the whole range of a. It was thought that increased coupling strength may result in more interesting behavior in these directions. The results in Figures 5.17–5.19 are very similar to the original symmetric case for the negative scan. Period–doubling is the route to chaos, and the chaotic region is interrupted by a stable period–2 solution. These transitions happen for slightly more positive values of a. The maximal Lyapunov exponent associated with this chaotic region is higher than the original, having a value of over 0.09.

The period–2 solution occurring post–chaos is much more stable than the original symmetric case, occurring over a range more than four times as large. Inspection of the phase projections shows that this period–2 solution is not the same that occurs in the original system. The current system also differs with an additional jump occurring at $a = -7.24$. The phase projection in the x_3–x_4 plane shows the jump is due to the

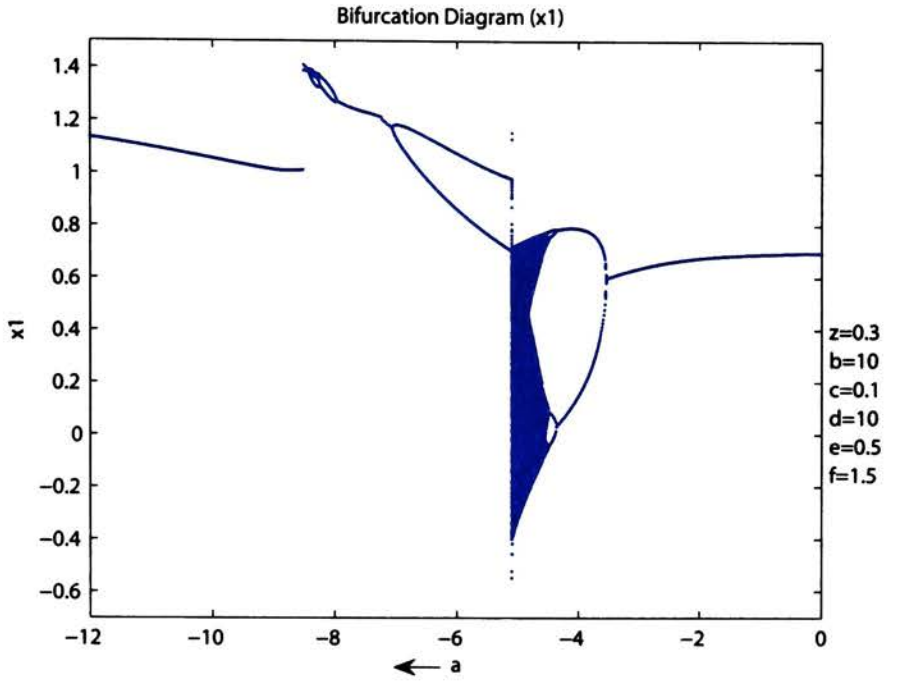


Figure 5.17: Bifurcation diagram for negative a scan with increased coupling strength (x_1)

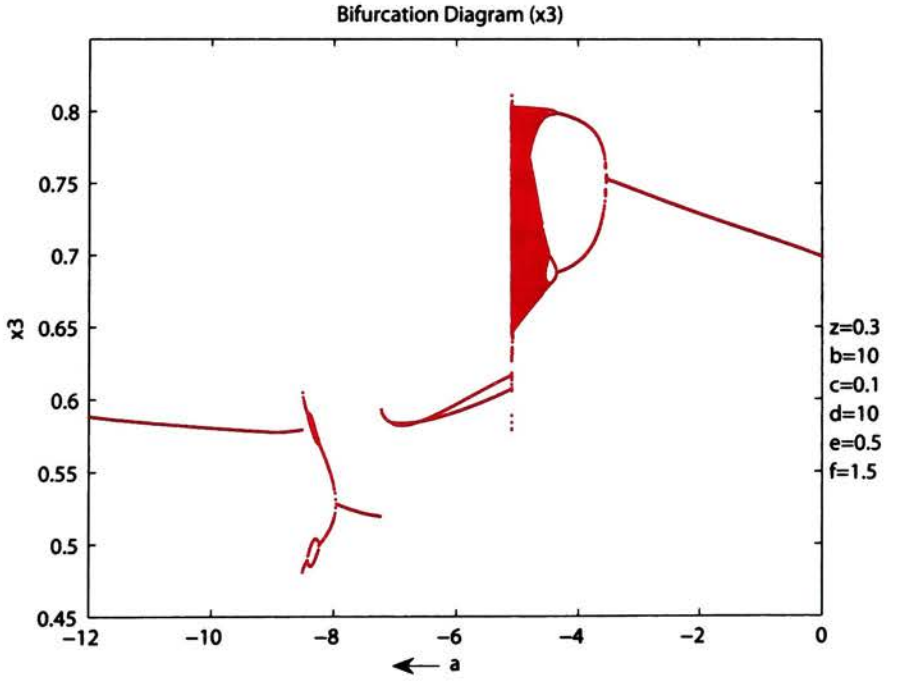


Figure 5.18: Bifurcation diagram for negative a scan with increased coupling strength (x_3)

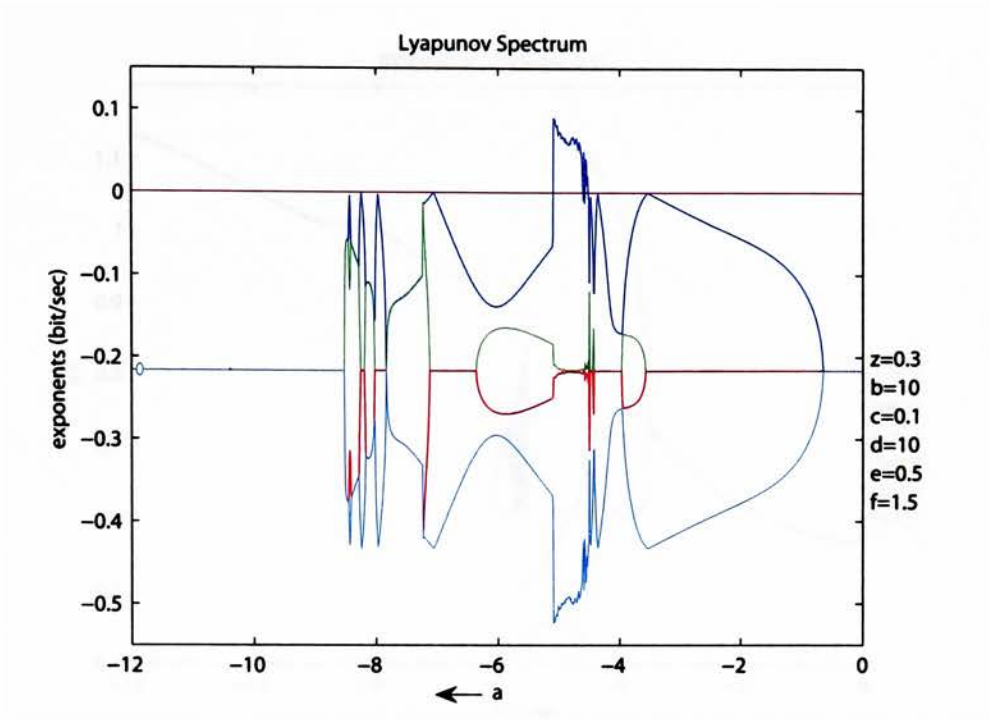


Figure 5.19: Lyapunov spectrum for negative a scan with increased coupling strength

sudden attraction of an orbit that is the mirror–symmetric version of the current period–1 solution. A similar period–doubling sequence near $a = -8$ occurs in this system. Unlike the original, the sequence results in a period–4 solution before it is forced to period–2 and the jump into the low amplitude low power cycle due to well entrainment. The solution is also entrained in the opposite well as the original system, which is probably the result of the additional jump that occurred when the mirror–symmetric period–1 solution became more attracting. The jump into the low amplitude cycles is again associated with a 20 dB reduction in power in the x_1 signal.

The positive scan over the range from -12 to 0 results in dramatically different behaviors than the negative scan, the most notable of which is the additional chaotic region encountered near $a = -6$ (see Figures 5.20–5.22).

The well–entrained cycle grows in amplitude. A jump takes place before the solution undergoes a period-doubling sequence to chaos. Unlike the negative scan, this chaotic region occurs in only one well and not over both (see Figure 5.23). The chaotic region is interrupted by a period-2 solution similar in shape to the period-2 solution that occurred

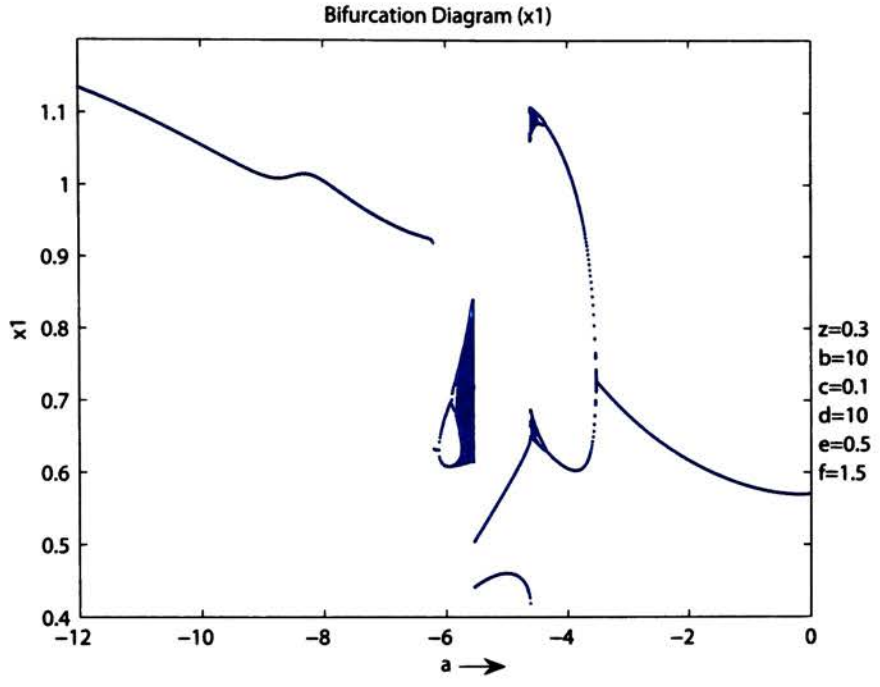


Figure 5.20: Bifurcation diagram for positive a scan with increased coupling strength (x_1)

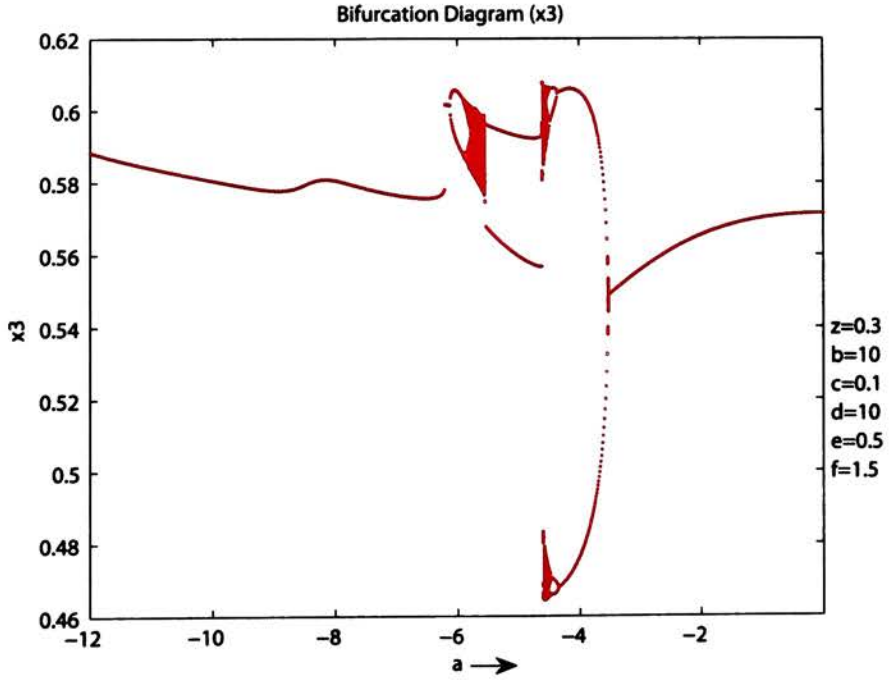


Figure 5.21: Bifurcation diagram for positive a scan with increased coupling strength (x_3)

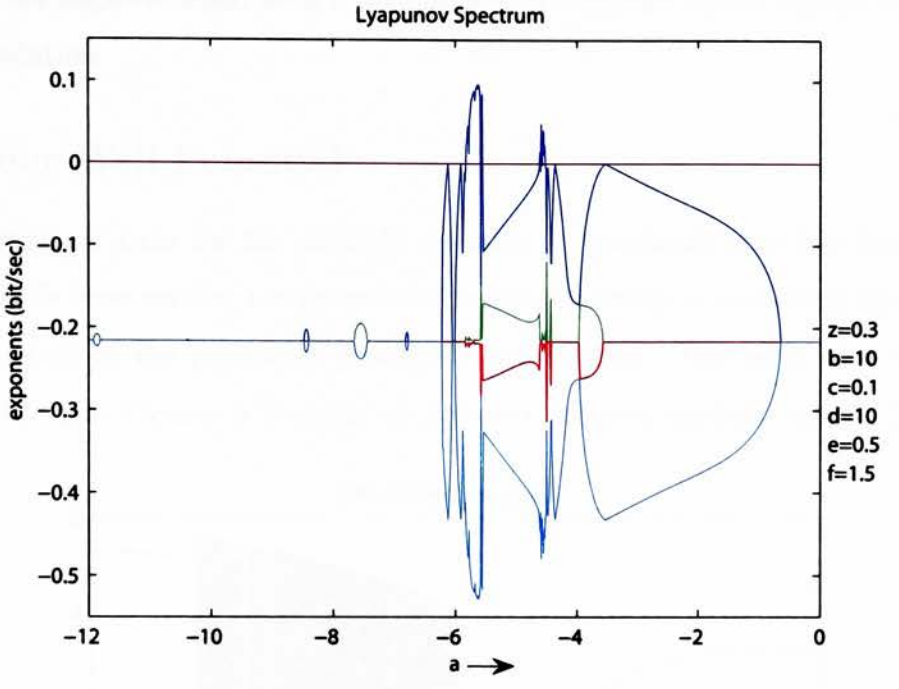
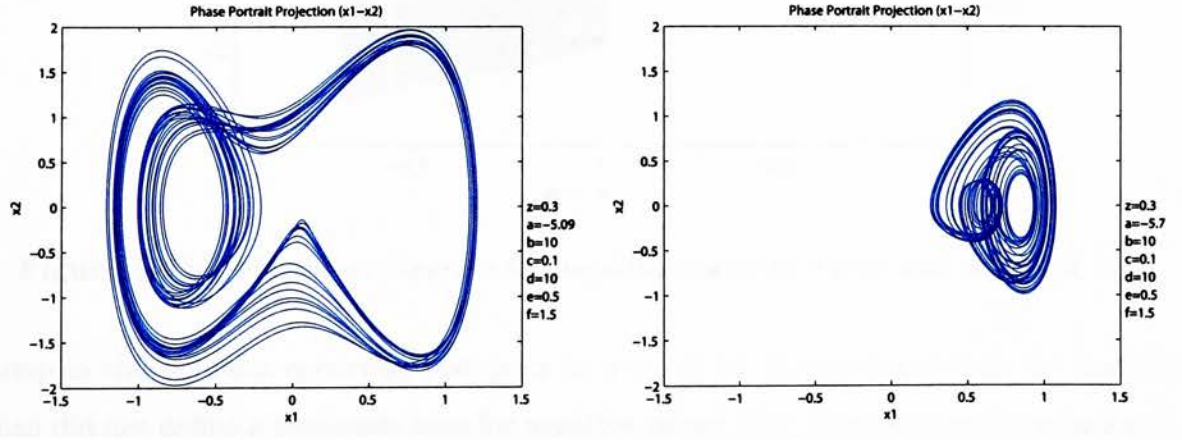


Figure 5.22: Lyapunov spectrum for positive a scan with increased coupling strength



(a) **(b)**

Figure 5.23: Phase portrait projections in the x_1–x_2 plane for a scan with increased coupling strength that correspond to: (b) negative scan with two well chaos, $a = -5.09$; (b) positive scan with one well chaos, $a = -5.7$.

post chaos in the original symmetric system. The solution path then follows the same behavior as the negative scan, with a period-halving sequence out of chaotic behavior to a period-1 solution.

5.2.4 Four Well Potential

The final group of scans for the variable a results in potentials that has four separate wells. The wells have smaller nonlinear components resulting in wells that are relatively "deep" compared to the potentials previously investigated. The value of a was varied from 0 to -10. See Figures 5.24–5.26. A period-1 solution initially exists. There is a

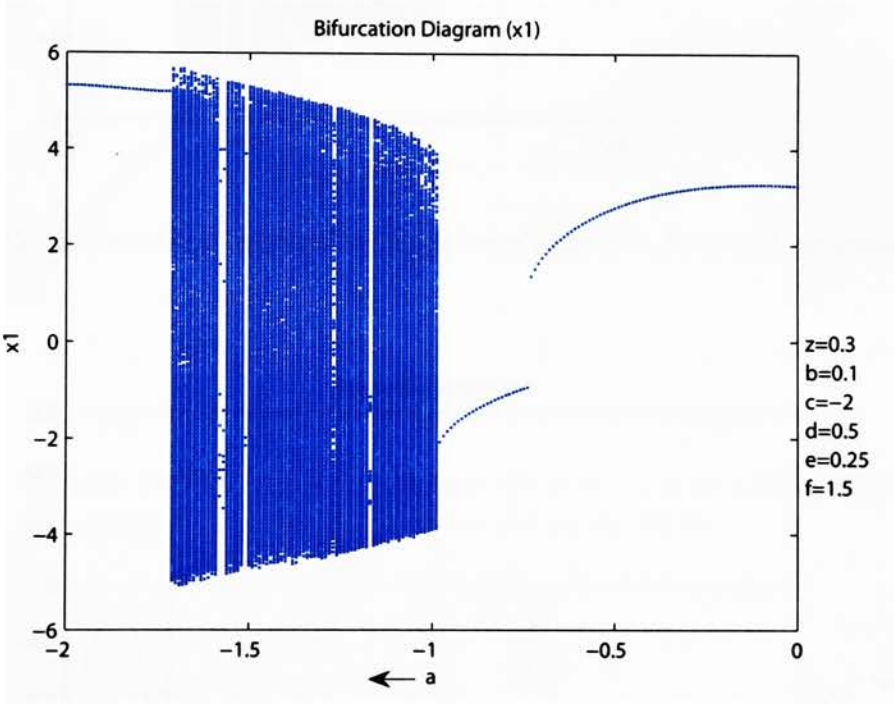


Figure 5.24: Bifurcation diagram for negative a scan in a four well potential (x_1)

jump in this solution occurring just prior to $a = -0.74$. It was found that the positive scan did not define a hysteresis loop for negative values of a. Crisis causes the solution to become chaotic. The chaotic region persists from -0.99 to -1.72. The maximal Lyapunov exponent associated with this chaotic region is approximately 0.25, resulting in the most chaotic region thus far. As observed in the Poincaré maps in Figure 5.27, the folding characteristics of the chaotic attractor cause the invariant manifolds to become sufficiently

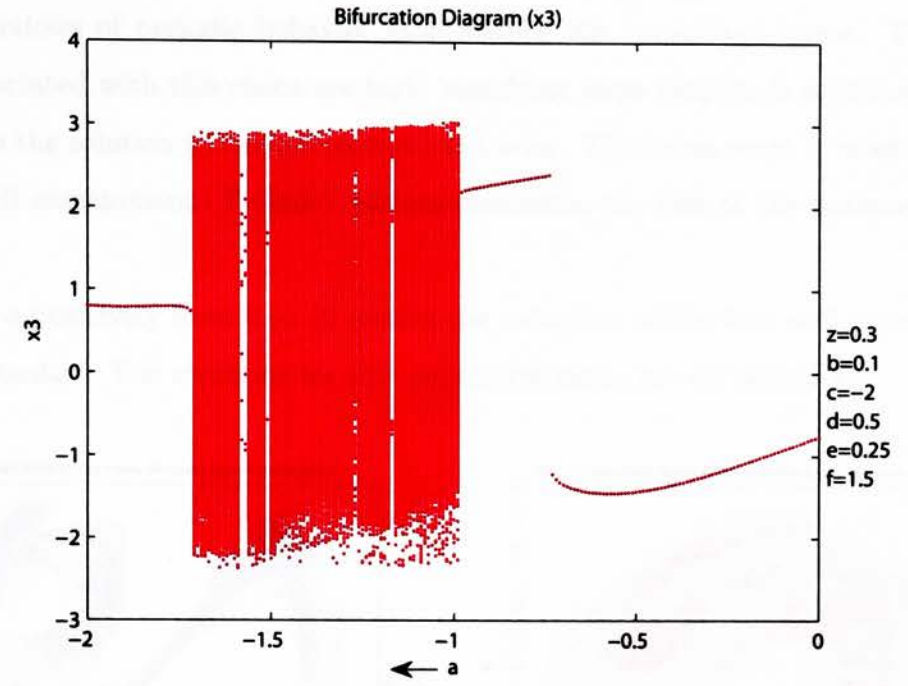


Figure 5.25: Bifurcation diagram for negative a scan in a four well potential (x_3)

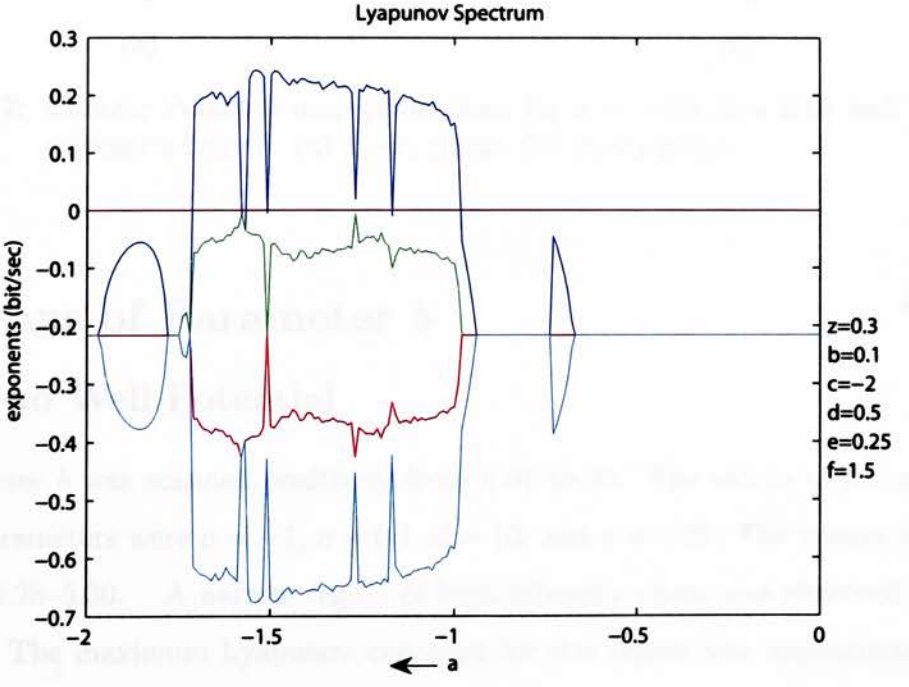
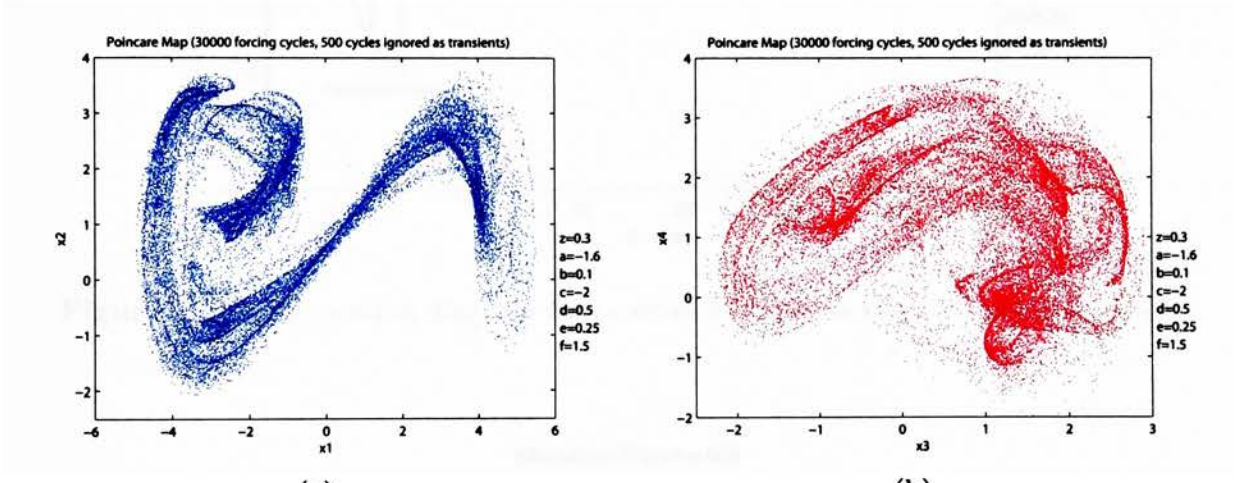


Figure 5.26: Lyapunov spectrum for negative a scan in a four well potential

tangled such that the self-similar fractal structure becomes difficult to observe.

Brief windows of periodic behavior exist within this broadband chaos. The power spectra associated with this chaos are high, signifying large amplitude motions. A crisis event causes the solution to resume period-1 behavior. This crisis event is most likely the result of well entrainment. Period-1 motion dominates the rest of the parameter space in this scan.

Varying a positively from 0 to 10 results the reduction of the four well potential to a two well potential. The solutions for this parameter range are all period-1.



(a) (b)

Figure 5.27: Chaotic Poincaré map projections for $a = -1.6$ in a four well potential corresponding to: (a) x_1–x_2 plane; (b) x_3–x_4 plane.

5.3 Scans of Parameter b

5.3.1 Two Well Potential

The parameter b was scanned positively from 0.01 to 35. The values of the remaining equation parameters were $a = -1$, $c = 0.1$, $d = 10$, and $e = 0.25$. The results are shown in Figures 5.28–5.30. A narrow region of high intensity chaos was observed for small values of b. The maximum Lyapunov exponent for this region was approximately 0.25. The chaotic region is ended by a crisis event rendering the solution period-1. A jump in this solution occurs when $b = 5.89$. The solution remains period-1 until a period-doubling

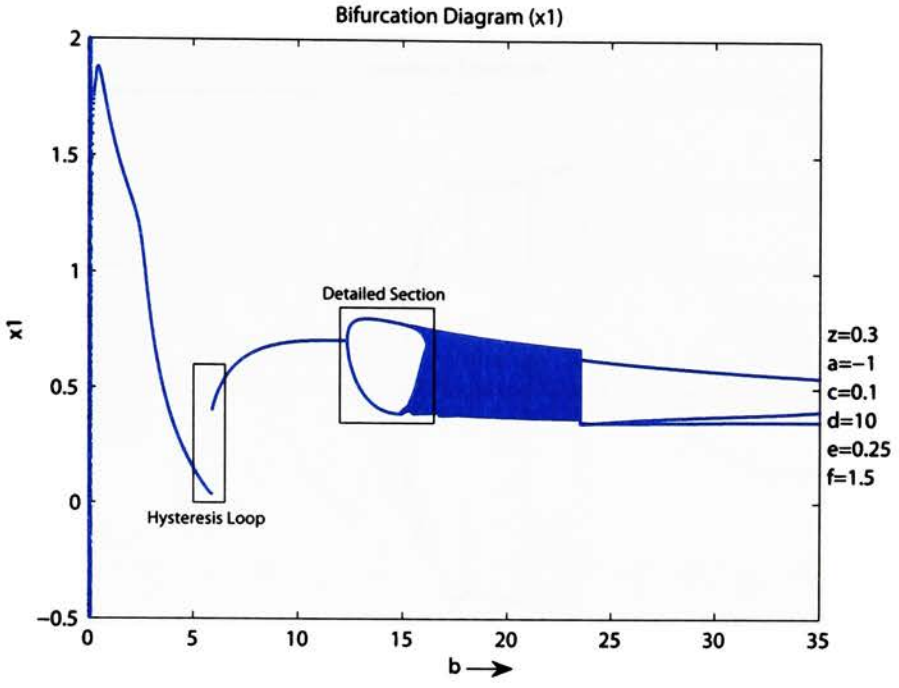


Figure 5.28: Bifurcation diagram for positive b scan in a two well potential (x_1)

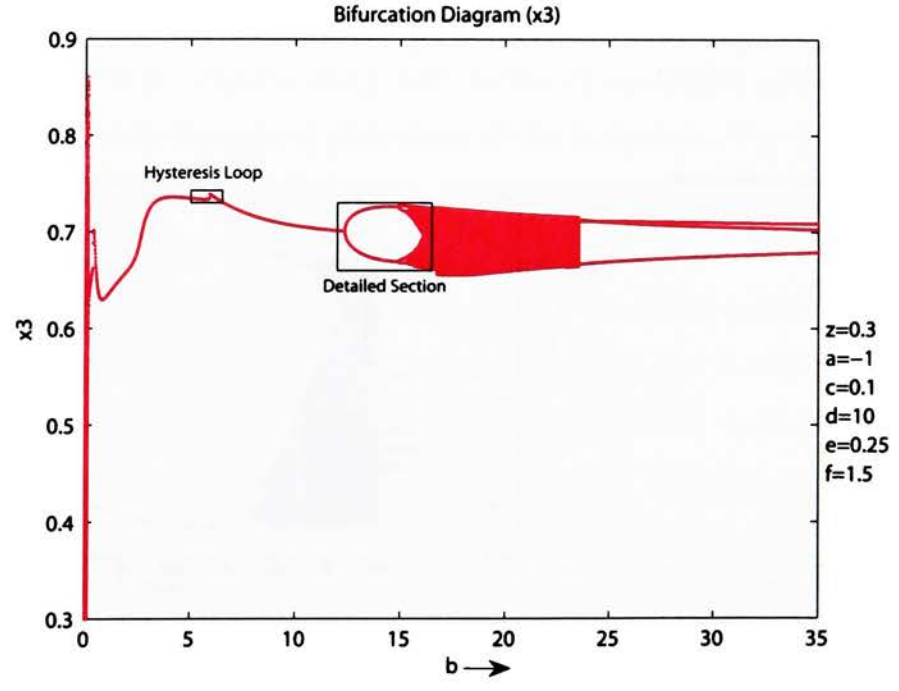


Figure 5.29: Bifurcation diagram for positive b scan in a two well potential (x_3)

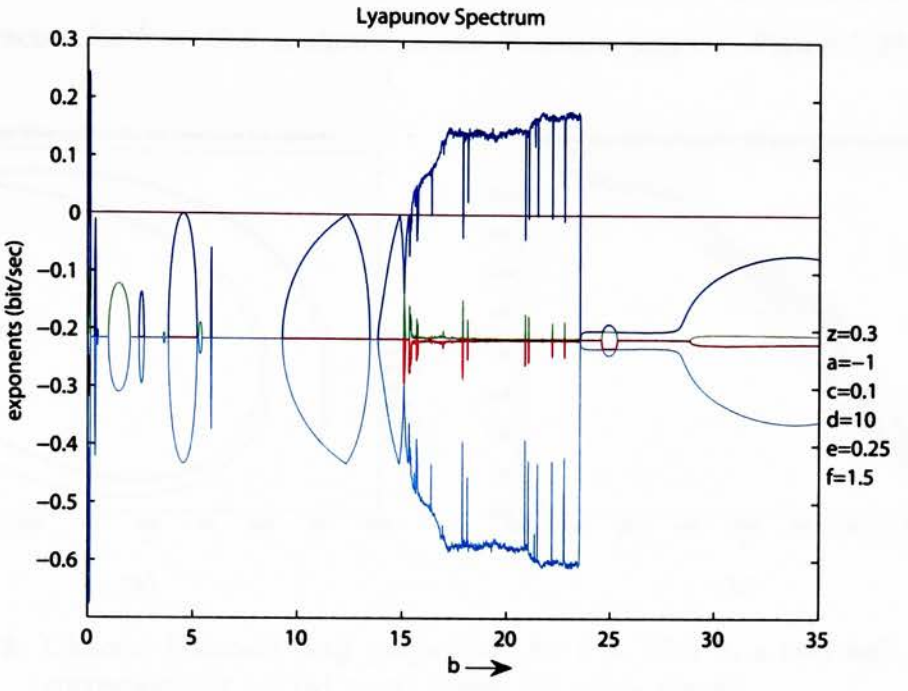
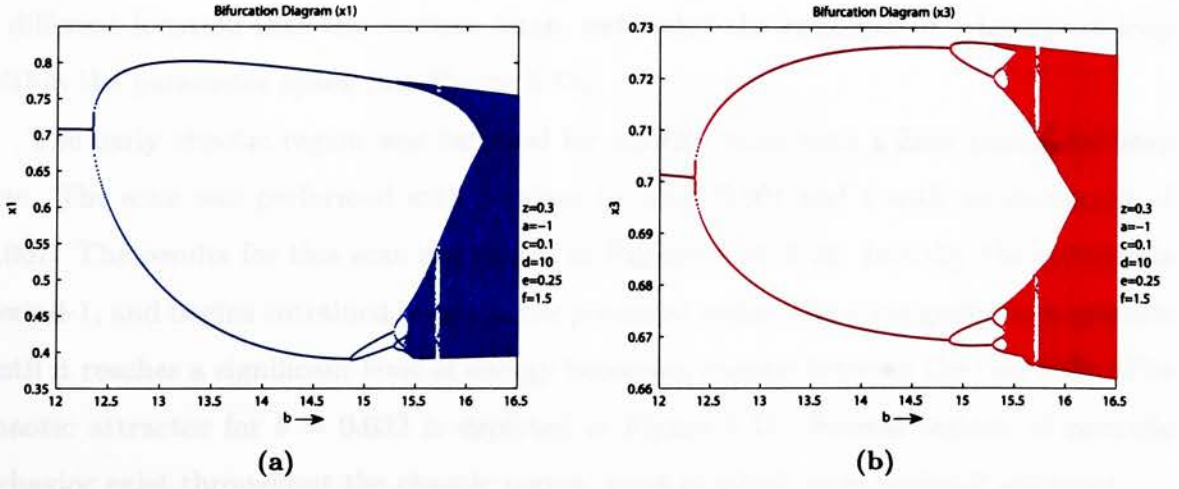


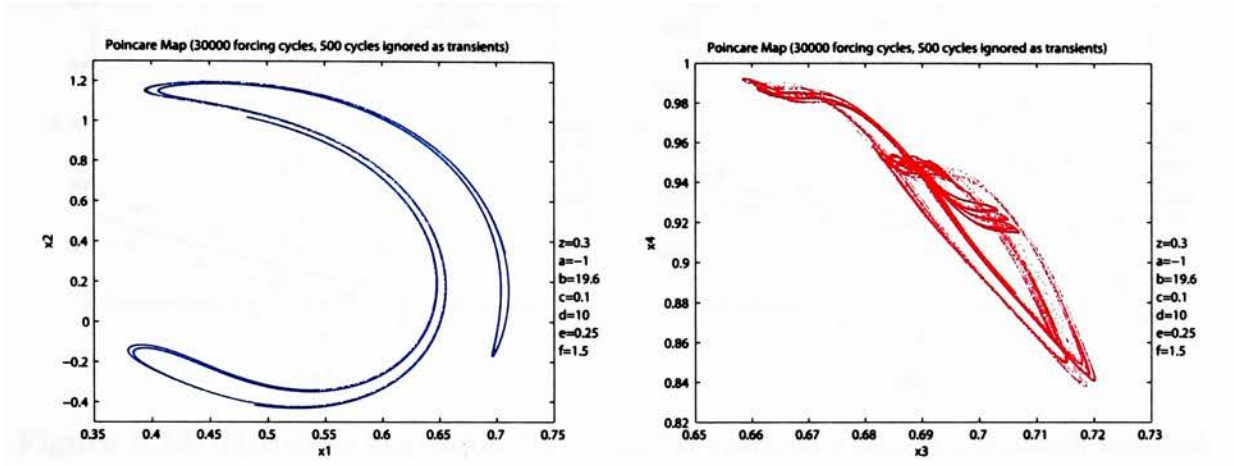
Figure 5.30: Lyapunov spectrum for positive b scan in a two well potential



(a) **(b)**

Figure 5.31: Period–doubling cascade for b scan in a two well potential

cascade to chaos begins at $b = 12.34$. The attractor becomes fully developed 16.16. This chaotic region is extremely stable and persists in the parameter space until 23.52. The chaotic attractor for $b = 19.6$ is shown in the Poincaré maps in Figure 5.32. Several



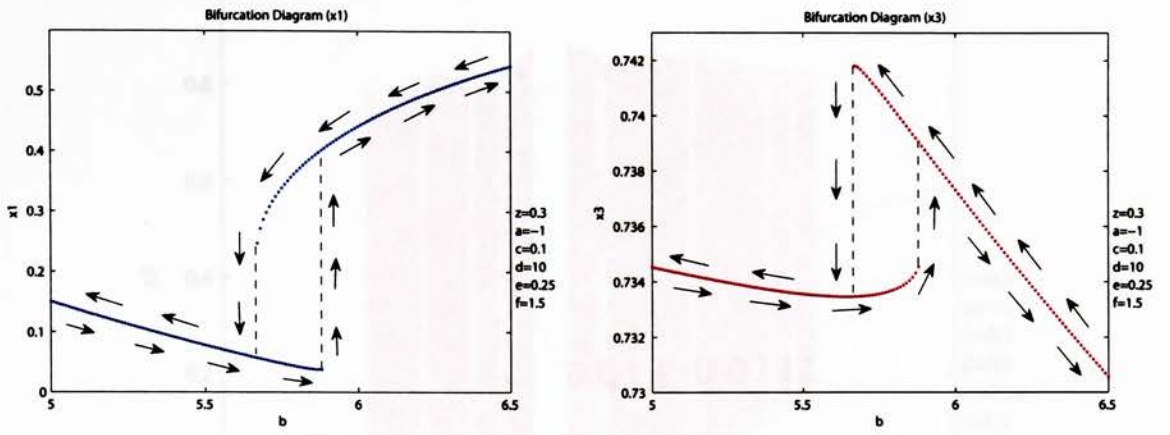
(a) (b)

Figure 5.32: Chaotic Poincaré map projections for $b = 19.6$ in a two well potential corresponding to: (a) x_1–x_2 plane; (b) x_3–x_4 plane.

periodic windows exist throughout this chaotic band. A crisis event renders a period-3 solution stable until the end of the parameter scan.

A negative scan from 10 to 2.2 with a -0.01 decrement resulted in a jump of the period-1 solution that persists throughout that range of the parameter. This jump occurred in a different location that the positive jump, indicating the existence of a hysteresis loop within the parameter space (see Figure 5.33).

The early chaotic region was targeted for another scan with a finer parameter step size. The scan was performed with b values between 0.001 and 1 with an increment of 0.001. The results for this scan are shown in Figures 5.34–5.36. Initially the solution is period-1, and begins entrained in one of the potential wells. The cycle grows in amplitude until it reaches a significant level of energy becoming chaotic between the two wells. The chaotic attractor for $b = 0.032$ is depicted in Figure 5.37. Several regions of periodic behavior exist throughout the chaotic region, most of which were period-3 solutions.



(a) (b)

Figure 5.33: Hysteresis loop through b parameter space for b scan in a two well potential

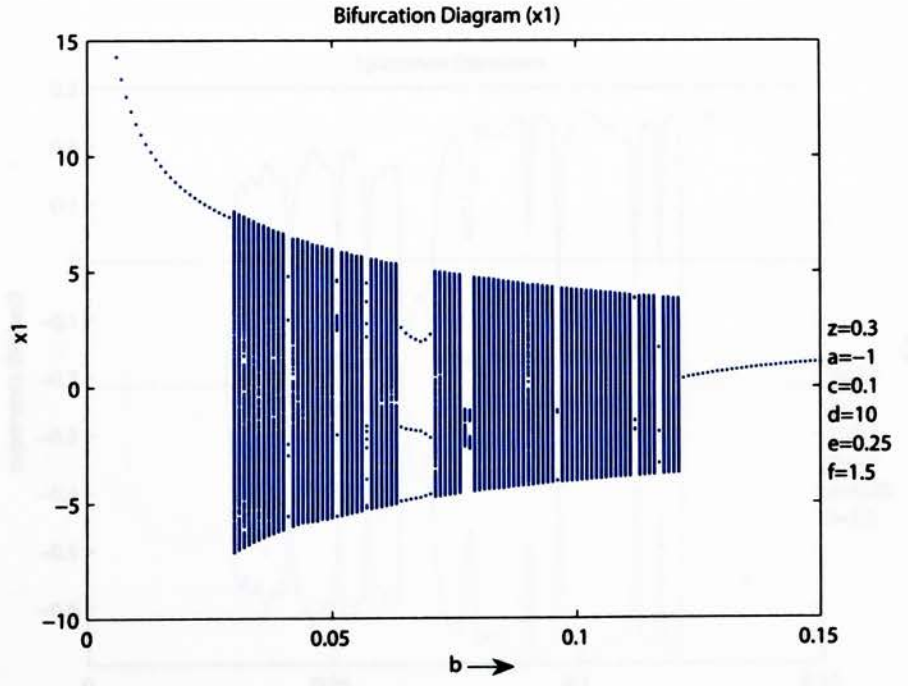


Figure 5.34: Bifurcation diagram for early chaotic b scan in a two well potential (x_1)

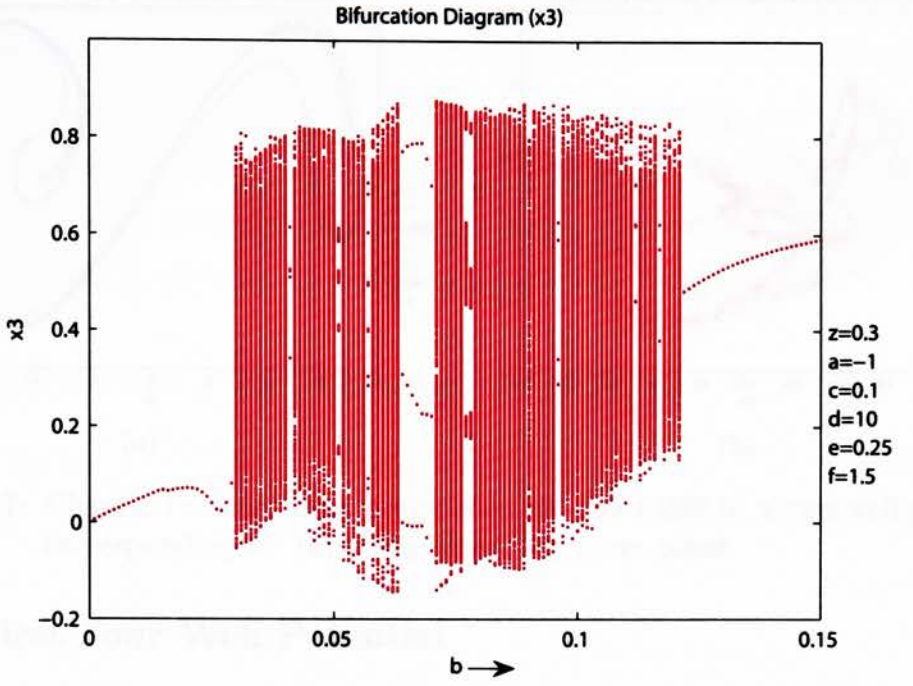


Figure 5.35: Bifurcation diagram for early chaotic b scan in a two well potential (x_3)

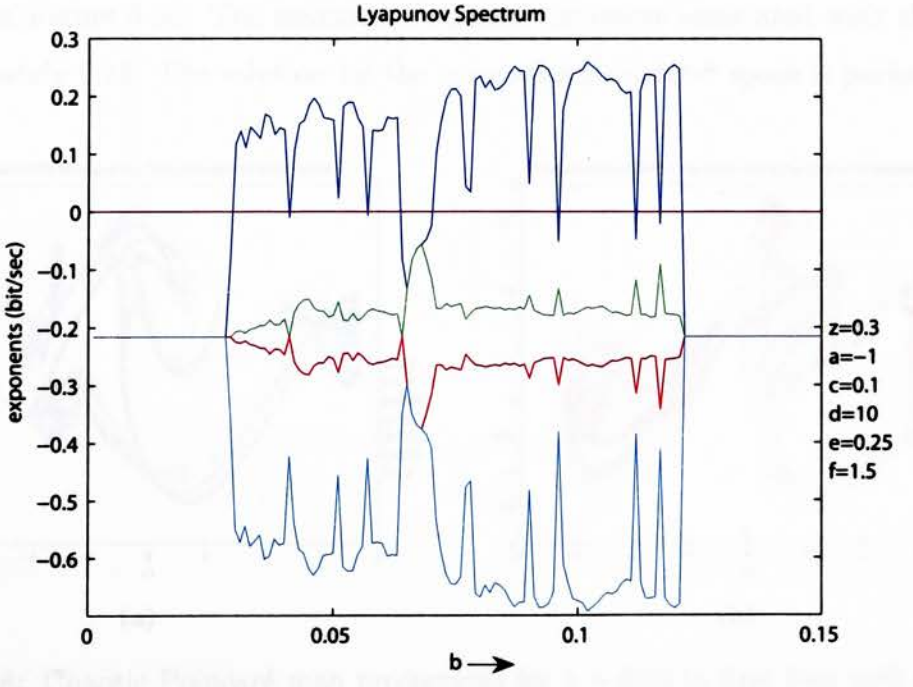
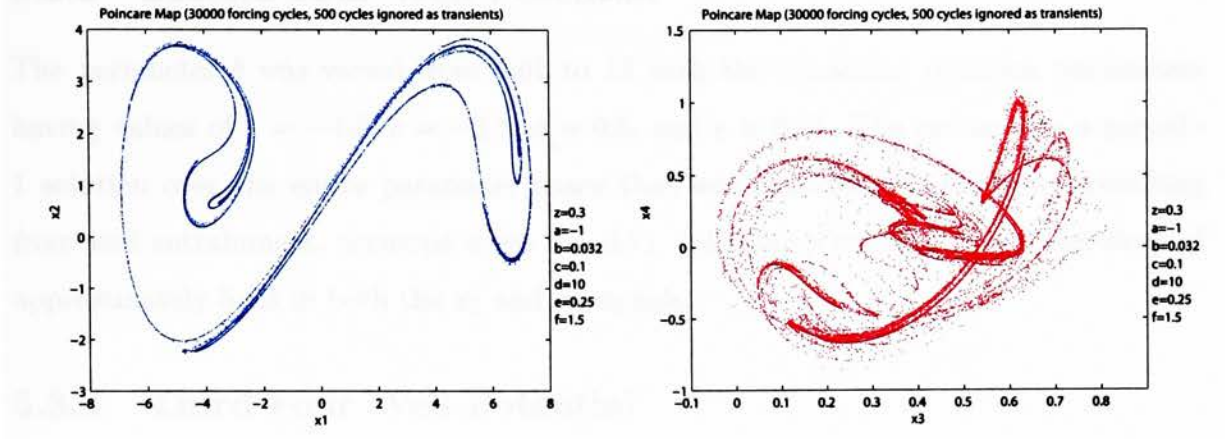


Figure 5.36: Lyapunov spectrum for early chaotic b scan in a two well potential

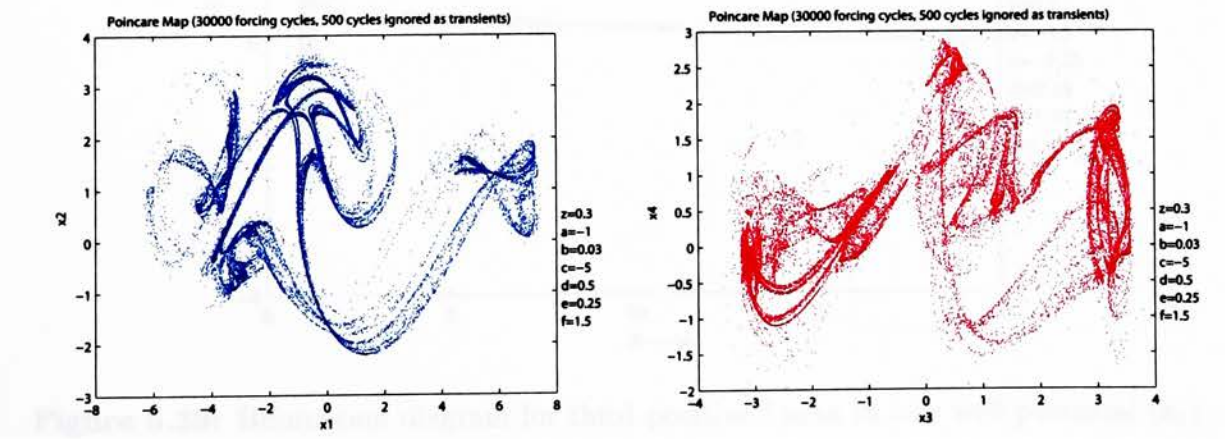


(a) (b)

Figure 5.37: Chaotic Poincaré map projections for $b = 0.032$ in a two well potential corresponding to: (a) x_1–x_2 plane; (b) x_3–x_4 plane.

5.3.2 First Four Well Potential

For this parameter scan, b was varied from 0.01 to 16 with the remaining system parameters values of $a = -1$, $c = -5$, $d = 0.5$, and $e = 0.25$. Chaos exists very briefly for only one value of the parameter, $b = 0.03$. The Poincaré maps for this parameter value are shown in Figure 5.38. The maximal Lyapunov exponent associated with this chaos is approximately 0.15. The solution for the remaining parameter space is period–1.



(a) (b)

Figure 5.38: Chaotic Poincaré map projections for $b = 0.03$ in first four well potential corresponding to: (a) x_1–x_2 plane; (b) x_3–x_4 plane.

5.3.3 Second Four Well Potential

The parameter b was varied from 0.01 to 12 with the remaining equation parameters having values of $a = -4.5$, $c = -3.5$, $d = 0.5$, and $e = 0.25$. The system has a period–1 solution over the entire parameter space that was considered. One jump, resulting from well entrainment, occurred when $b = 0.71$. This jump resulted in a power drop of approximately 5 dB in both the x_1 and x_3 signals.

5.3.4 Third Four Well Potential

The third b parameter scan in a four well potential has variables that were very similar to the second. The value of c was changed to -2.25, with the remaining values the same. The behavior of the system (Figures 5.39–5.41) is significantly different that the previous system. Crisis causes the system to become chaotic at $b = 0.87$. This chaotic region has

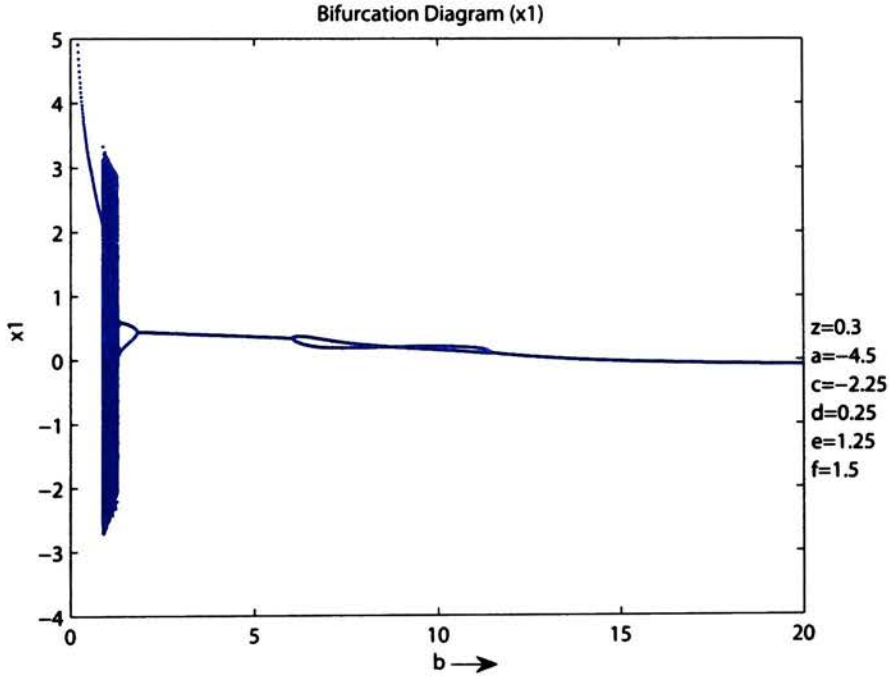


Figure 5.39: Bifurcation diagram for third positive b scan in four well potential (x_1)

the highest intensity observed in any of the parameter scans for the synchronous forcing condition. The maximal Lyapunov exponent is near 0.35. Unlike the majority of the chaotic regions observed in this study, this particular band of chaos has no periodic or

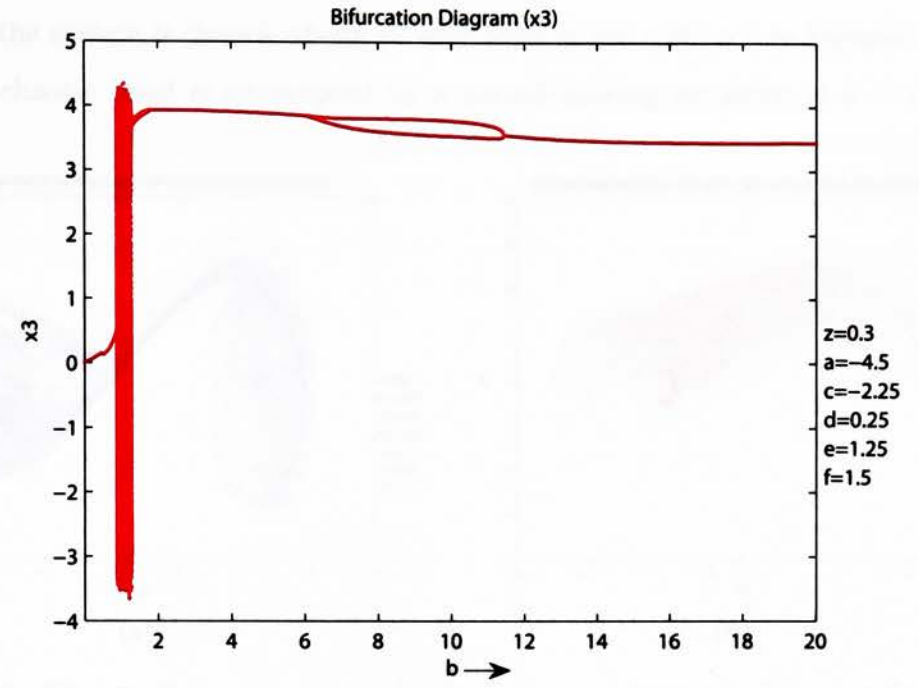


Figure 5.40: Bifurcation diagram for third positive b scan in four well potential (x_3)

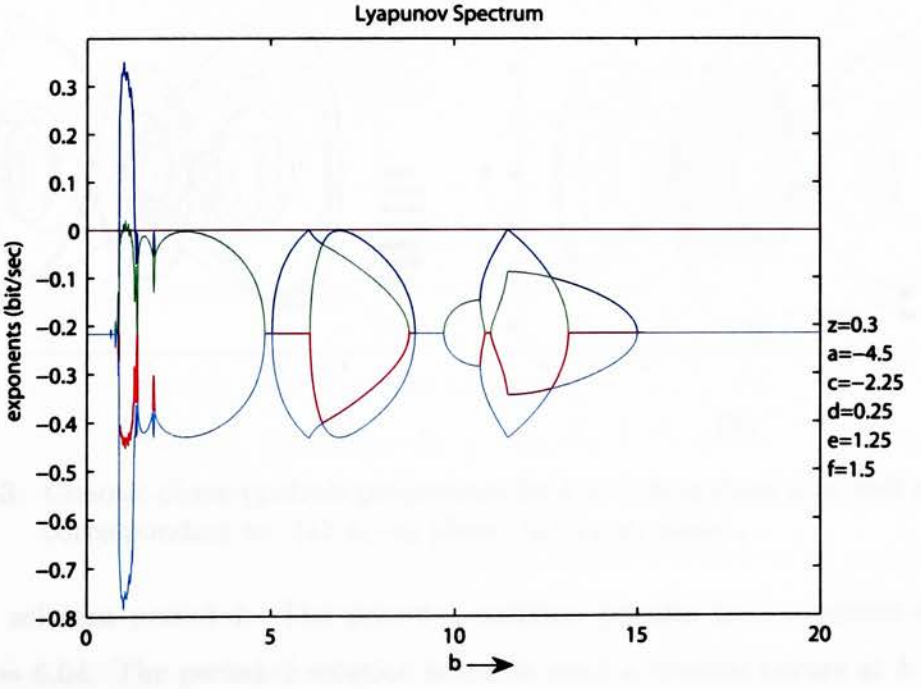
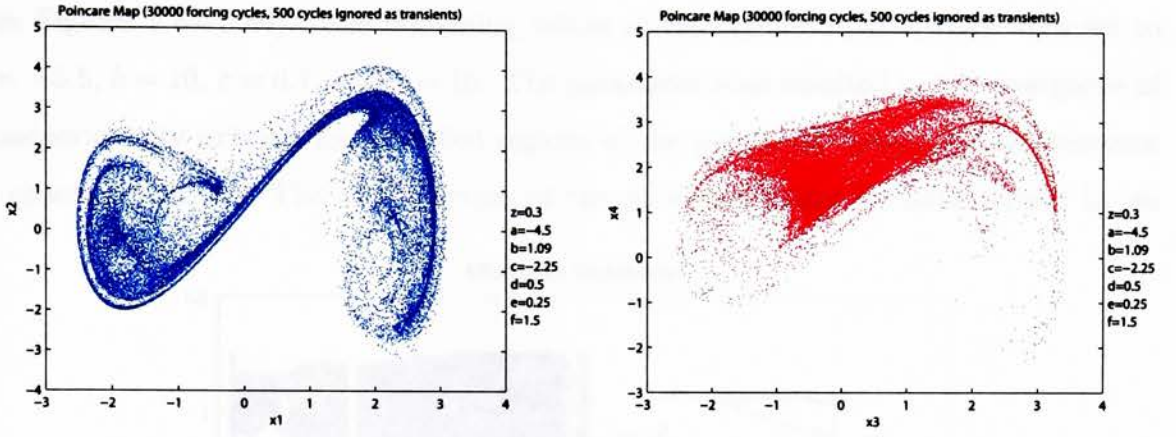


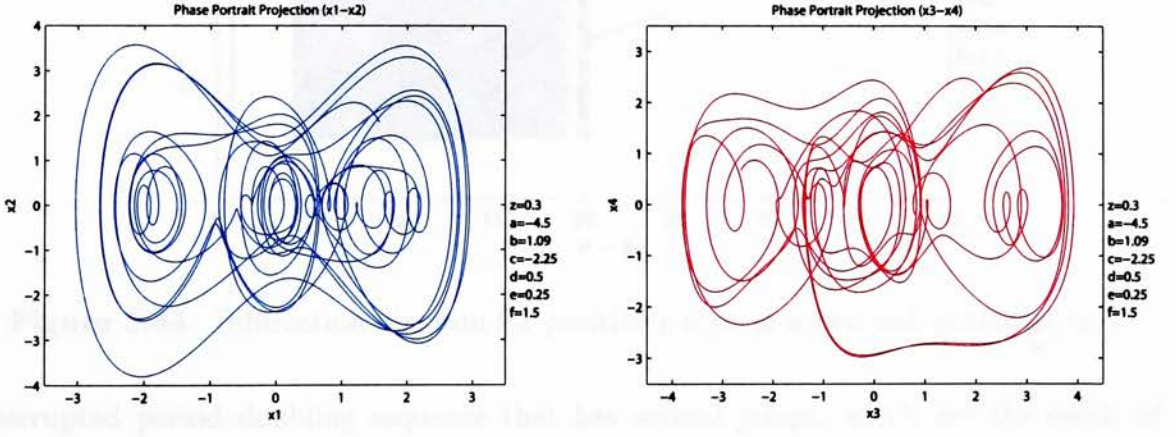
Figure 5.41: Lyapunov spectrum for third positive b scan in four well potential

quasiperiodic windows. Also, from observations of the Poincaré maps and phase portrait projections the system is chaotic about all four wells in the system (see Figures 5.42 and 5.43). The chaotic band is interrupted by a period–halving sequence at $b = 1.28$ that



(a) (b)

Figure 5.42: Chaotic Poincaré map projections for $b = 1.09$ in third four well potential corresponding to: (a) x_1–x_2 plane; (b) x_3–x_4 plane.



(a) (b)

Figure 5.43: Chaotic phase portrait projections for $b = 1.09$ in third four well potential corresponding to: (a) x_1–x_2 plane; (b) x_3–x_4 plane.

renders the solution period–1. The period–1 solution persists until a period–doubling occurs at $b = 6.04$. The period–2 solution is stable until a reversal occurs at $b = 11.45$. The resulting period–1 solution is stable for the remaining parameter space considered in this scan.

5.4 Scans of Parameter e

5.4.1 Two Well Potential

Varying e from 0 to 40 in a two well potential results in a variety of interesting behavior (see Figures 5.44–5.46). The remaining values of the equation parameters were set to $a = -5.5$, $b = 10$, $c = 0.1$, and $d = 10$. The parameter scan resulted in the emergence of quasiperiodicity existing for extended regions of the parameter space, and as precursor to chaotic behavior. The early portion of the parameter space is characterized by an

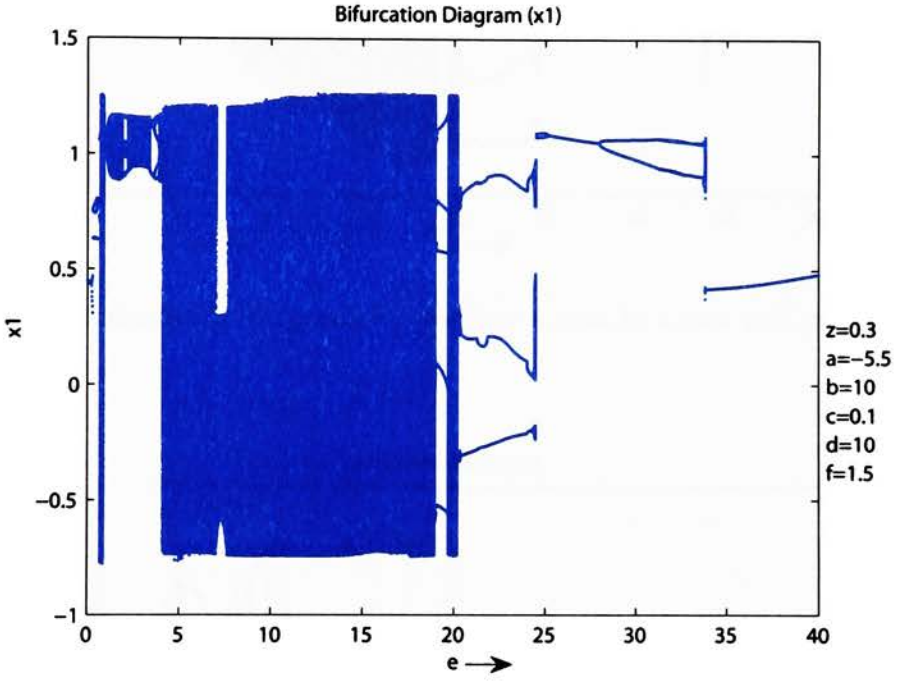


Figure 5.44: Bifurcation diagram for positive e scan in a two well potential (x_1)

interrupted period–doubling sequence that has several jumps, which are the result of competing well entrainments. This sequence is interrupted at period–4 and reverts to a period–2 solution before a crisis event signifies the onset of chaotic behavior. The early chaotic region exists from 0.72 to 0.88, and another crisis event causes the system to exit the chaotic attractor and exhibit period–1 behavior. The period–1 behavior is stable until the onset of an extended region of quasiperiodicity that begins when $e = 1.02$. This broadband region of quasiperiodicity extends until the onset of chaotic behavior at $e = 3.9$. The torus attractor for $e = 2.45$ is shown in Figure 5.47.

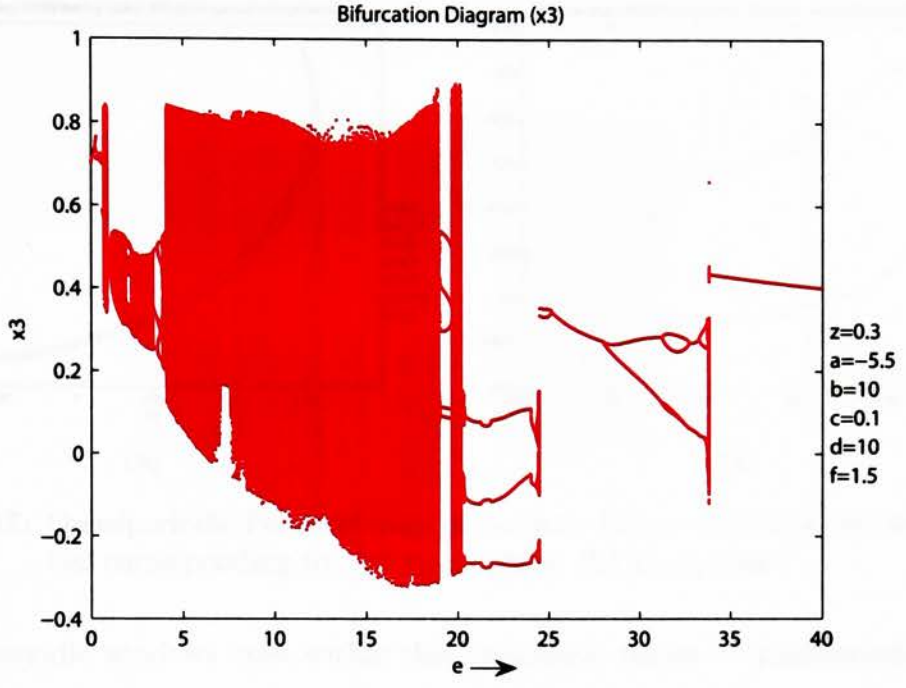


Figure 5.45: Bifurcation diagram for positive e scan in a two well potential (x_3)

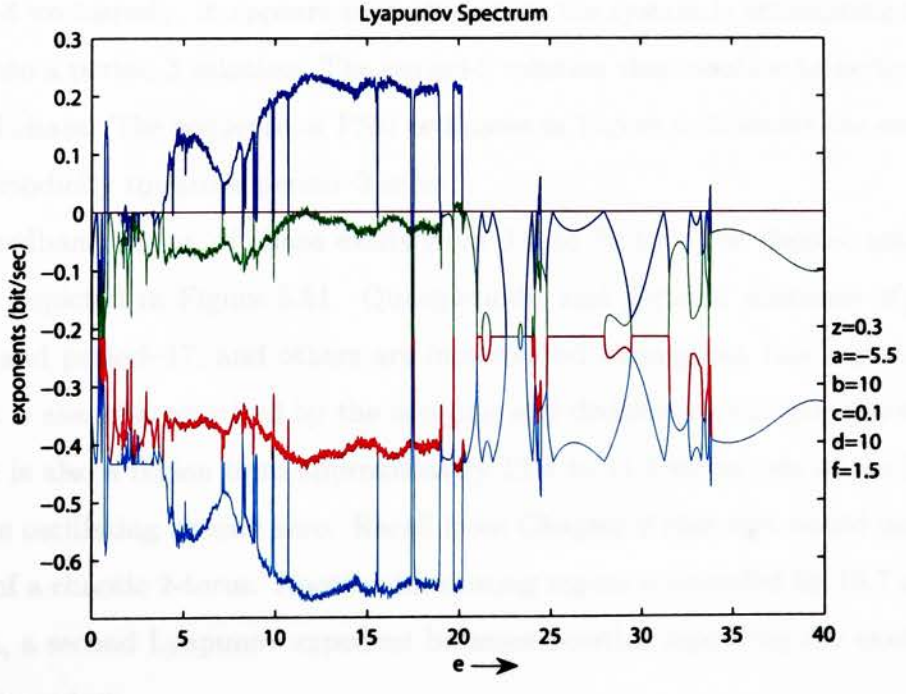
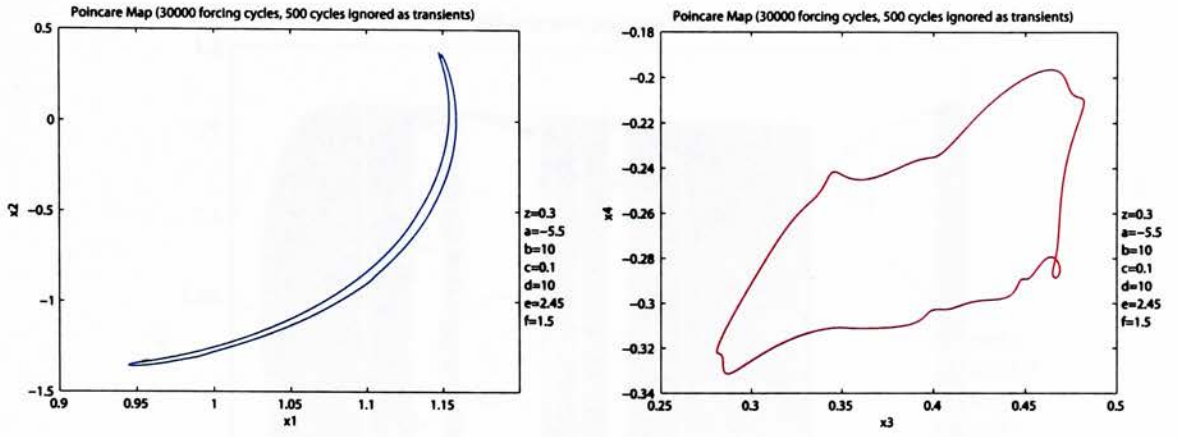


Figure 5.46: Lyapunov spectrum for positive e scan in a two well potential



(a) (b)

Figure 5.47: Quasiperiodic Poincaré map projections for $e = 2.45$ in a two well potential corresponding to: (a) x_1–x_2 plane; (b) x_3–x_4 plane.

Many periodic windows exist within this broadband region of quasiperiodicity. Detailed views showing the quasiperiodic bifurcation diagrams and Lyapunov spectrum from $e = 0.9$ to $e = 4$ are shown in Figures 5.48–5.50. Several of the larger regions include period-12, period-5, period-4 with a doubling to period-8 and halving back to period-4, and period-8 exclusively. It appears that ultimately the system is attempting to reorganize itself into a period-3 solution. The period-3 solution then doubles to period-6 before the onset of chaos. The sequence of PSD estimates in Figure 5.52 shows the evolution of the quasiperiodicity toward a period–3 orbit.

The broadband region of chaos exists from 3.9 to 20.18. The chaotic attractor for $e = 5.85$ is depicted in Figure 5.51. Quasiperiodic and periodic solutions of period–5, period–14, and period–17, and others are interspersed throughout this region of chaos. This region is also characterized by the merging and division of multiple chaotic attractors. There is also a region from approximately 11.4 to 11.8 where one of the Lyapunov exponents is oscillating around zero. Recall from Chapter 2 that this would indicate the possibility of a chaotic 2-torus. Another interesting region is bounded by 19.7 and 20.05. In this area, a second Lyapunov exponent becomes positive signifying the existence of a hyperchaotic system.

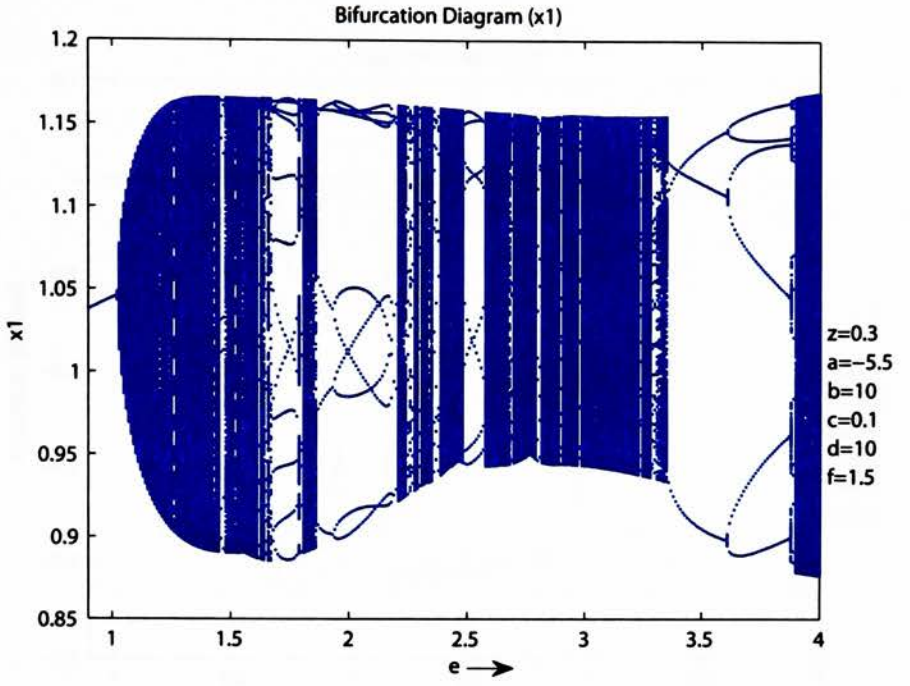


Figure 5.48: Bifurcation diagram for detailed section of two well potential e scan (x_1)

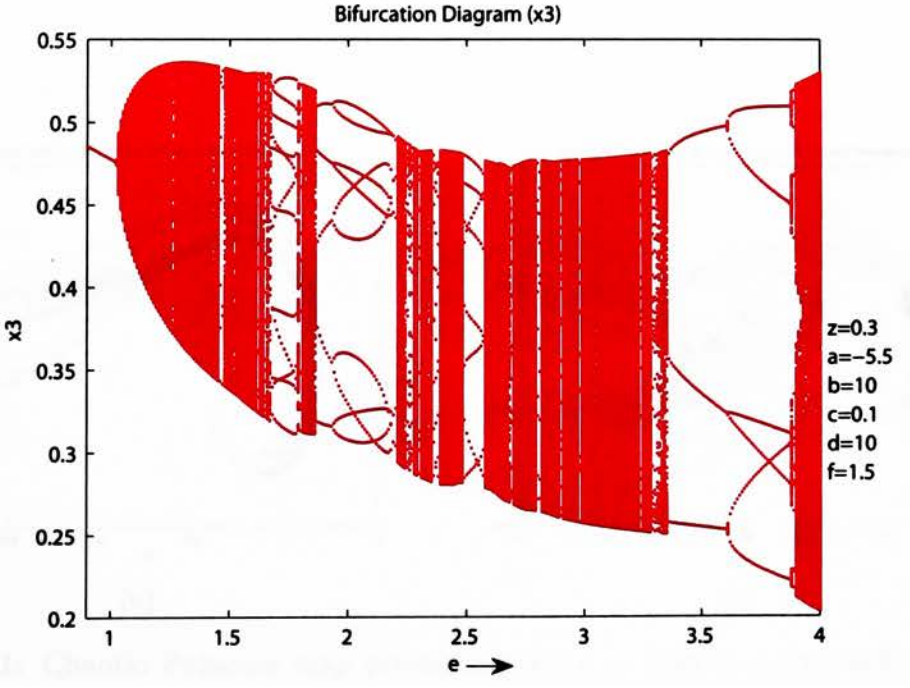


Figure 5.49: Bifurcation diagram for detailed section of two well potential e scan (x_3)

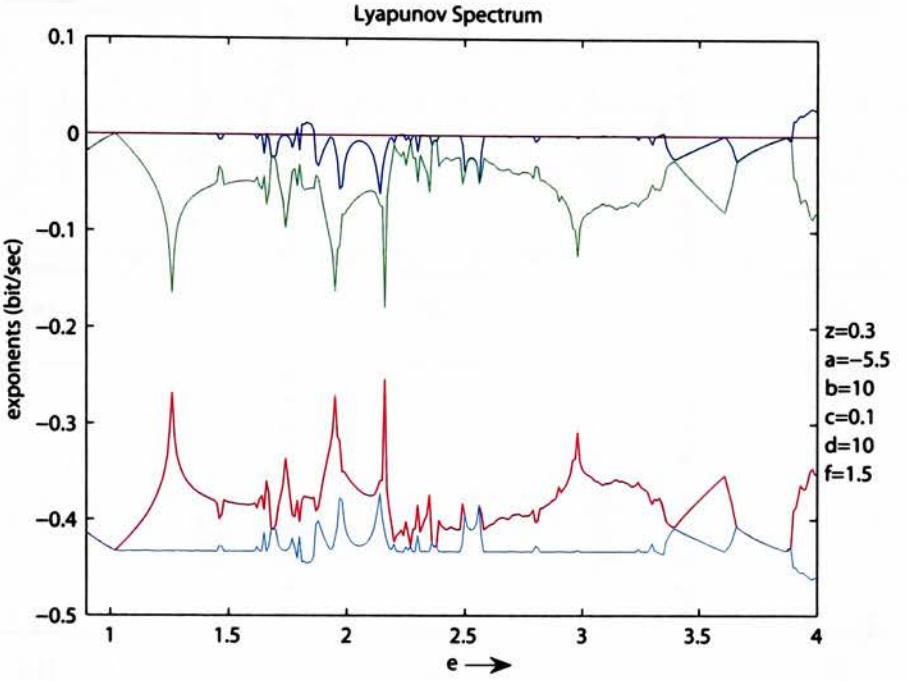
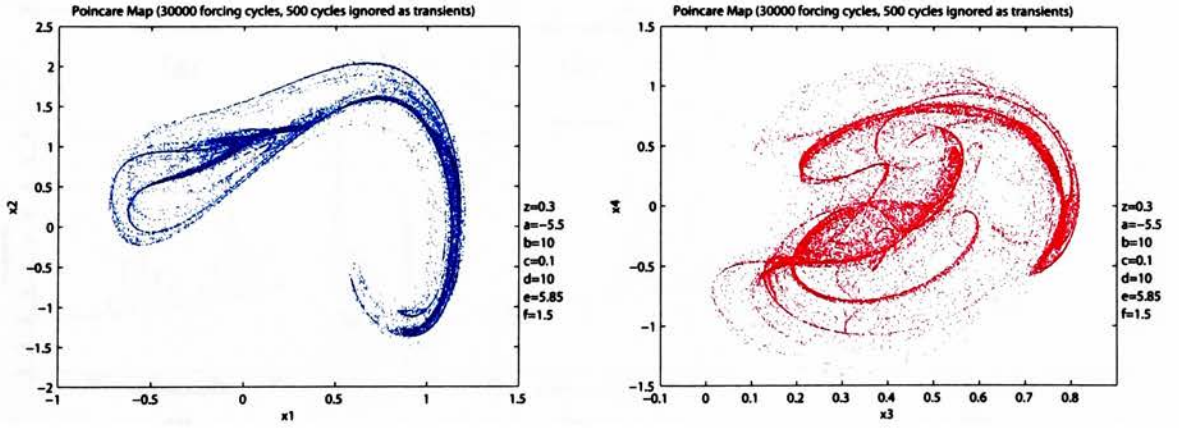


Figure 5.50: Lyapunov spectrum for detailed section of two well potential e scan



(a) **(b)**

Figure 5.51: Chaotic Poincaré map projections for $e = 5.85$ in a two well potential corresponding to: (a) x_1–x_2 plane; (b) x_3–x_4 plane.

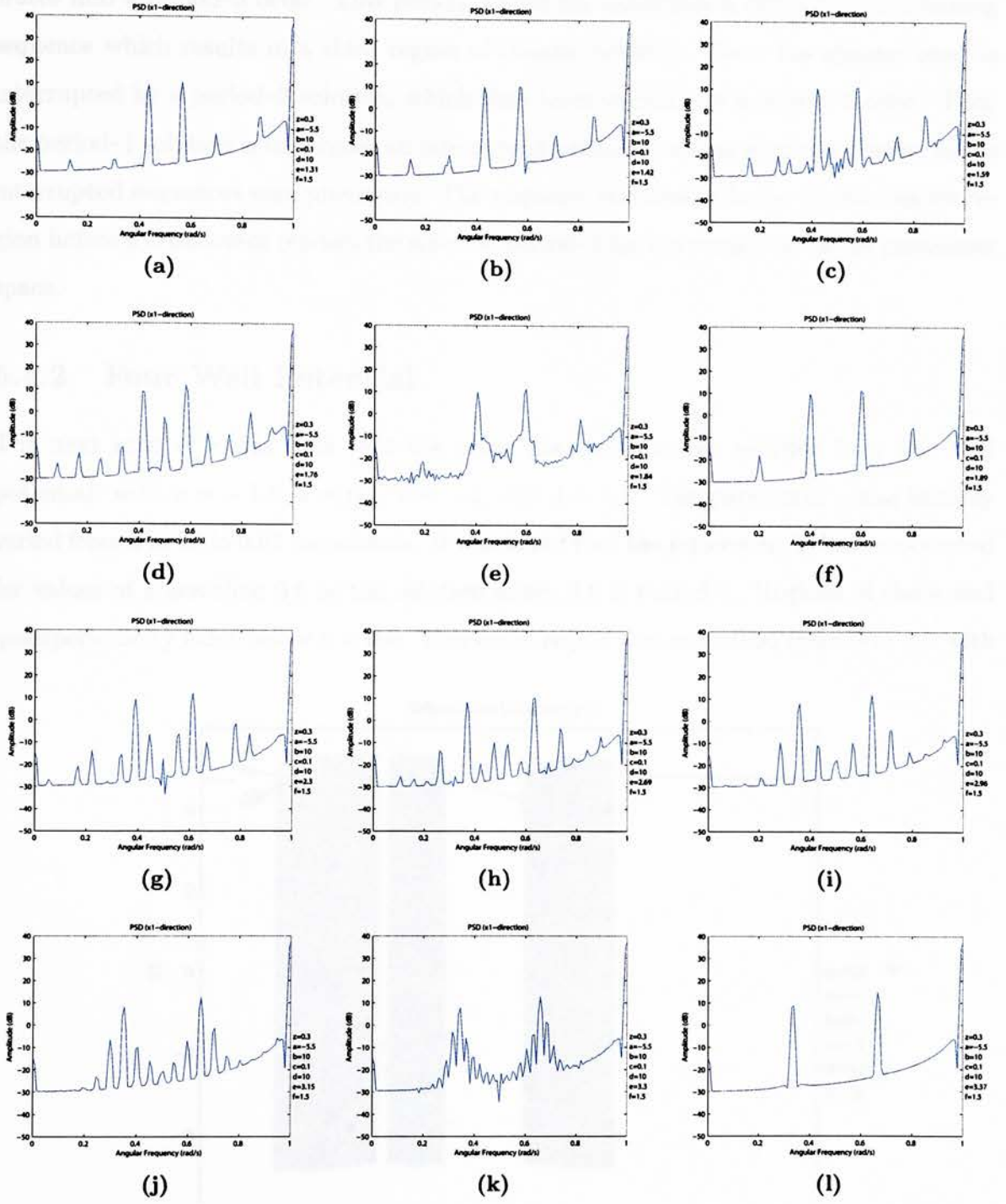


Figure 5.52: PSD estimates showing evolution toward a period–3 solution. e values correspond to: (a)1.31; (b)1.42; (c)1.59; (d)1.76; (e)1.84; (f)1.89; (g)2.3; (h)2.69; (i)2.96; (j)3.15; (k)3.3; and (l)3.37.

The chaotic region is finally interrupted by a small quasiperiodic solution that degenerates into a period–3 orbit. This period–3 solution undergoes a rapid period–doubling sequence which results in a third region of chaotic behavior. Next the chaotic band is interrupted by a period–2 solution, which then loses stability to a period–1 orbit. Then the period–1 solution is undergoes an interrupted period–doubling sequence similar to the interrupted sequences seen previously. The sequence culminates in the fourth chaotic region before a crisis event renders the solution period–1 for the remainder of the parameter space.

5.4.2 Four Well Potential

The next scan of e was such that the other parameter values resulted in a four well potential, with $a = -1.5$, $b = 0.1$, $c = -2$, and $d = 0.5$. The parameter e was initially varied from 0 to 20 in 0.01 increments. It was found that the interesting behavior occurred for values of e less than 0.6 as the solution above 0.6 is period–1. Regions of chaos and quasiperiodicity exist below $e = 0.6$. This early region was rescanned from 0 to 0.9 with

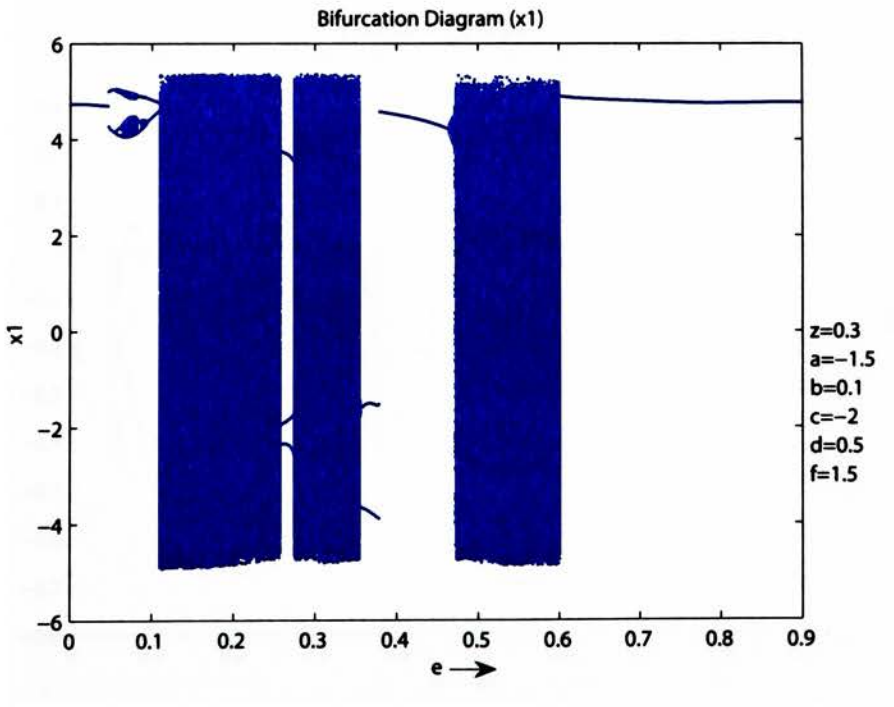


Figure 5.53: Bifurcation diagram for positive e scan in a four well potential (x_1)

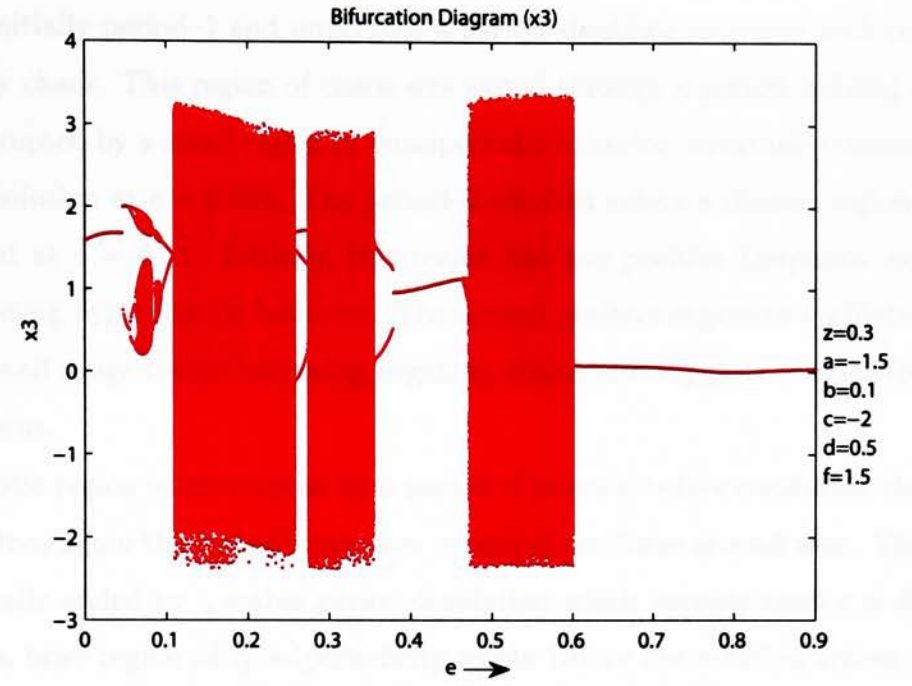


Figure 5.54: Bifurcation diagram for positive e scan in a four well potential (x_3)

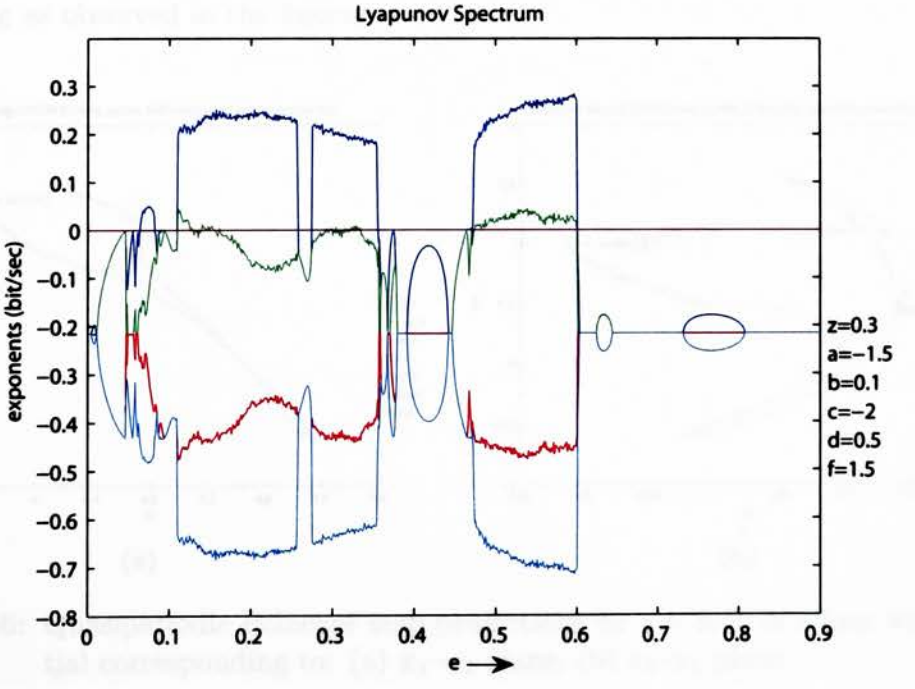
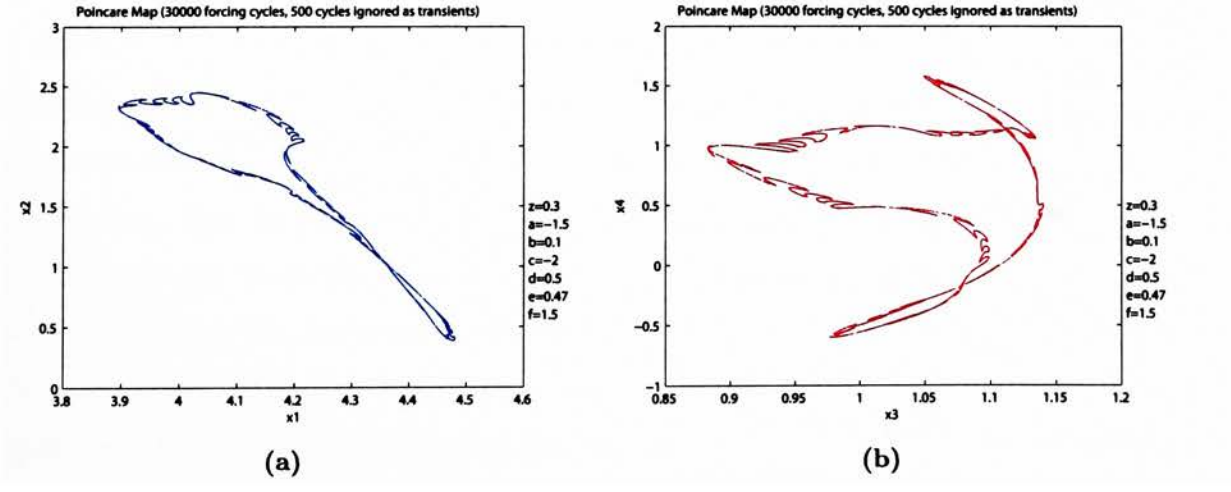


Figure 5.55: Lyapunov spectrum for positive e scan in a four well potential

a finer increment of 0.001, the results of which are shown in Figures 5.53–5.55. The solution is initially period–1 and undergoes a period–doubling sequence with resulted in low intensity chaos. This region of chaos was exited through a period–halving sequence that is interrupted by a small region of quasiperiodic behavior, eventually culminating in a period–2 solution at $e = 0.094$. The period–2 solution enters a chaotic region through a crisis event at $e = 0.11$. Initially, this region has two positive Lyapunov exponents, again suggesting hyperchaotic behavior. The second positive exponent oscillates around zero for a small range before becoming negative, which also suggests the possibility of a chaotic 2–torus.

The chaotic region is interrupted by a period–3 solution before reentering the chaotic attractor. Once again the second Lyapunov exponent oscillates around zero. The chaotic region is finally ended by a stable period–2 solution which persists until $e = 0.465$. At this point, a brief region of quasiperiodicity exists before the solution enters a second chaotic region. Throughout this region, two of the Lyapunov exponents are positive. Well entrainment ends this chaotic region. The torus attractor that precedes the second region of chaos is depicted in Figure 5.56. The torus is on the verge of being destroyed by wrinkling as observed in the figures.



(a) **(b)**

Figure 5.56: Quasiperiodic Poincaré map projections for $e = 0.47$ in a four well potential corresponding to: (a) x_1–x_2 plane; (b) x_3–x_4 plane.

5.5 Small Nonlinearities

Entries 20 to 25 and 29 to 35 in Table 5.1 represent scans that occurred in the realm of "small" nonlinearities. The purpose of these scans was to determine if critical values of the nonlinear components exist such that chaotic or quasiperiodic motions could result from small nonlinear contributions to the differential equations. The conclusion for all these scans is that the motion remained period–1, which suggests that a linearization of the system would probably be valid within these parameter ranges.

Chapter 6

Nonsynchronous System Analysis Results

Most of the parameter scans performed for the synchronous system were repeated for the nonsynchronous system. Thus Table 5.1, shows the parameter scans performed for the nonsynchronous system, with some minor additions and exceptions. The major differences are the inclusion or exclusion of scans that incorporated a finer parameter step size. In most of these cases, the nonsynchronous system did not exhibit the same rapid parameter–based variations in the dynamic behaviors over the same region of parameter space as the synchronous system. Scans with finer parameter step sizes were adjusted, added, or omitted accordingly.

Overall, it is difficult to anticipate what to expect for the nonsynchronous system given the synchronous system analysis results. On one hand, it is logical to expect that the nonsynchronous system will be more chaotic, based solely on the fact that it is more complicated. The forcing conditions are such that the sine wave being applied to the one oscillator will be at full amplitude when the cosine wave being applied to the other oscillator is zero. On the other hand, the lull in the forcing condition may allow the synchronous system to "drift" into more complex behavior. Also, the coincidence of the applied synchronous forcing lends to higher forcing energies, albeit periodically, than can be achieved by nonsynchronous forcing. Nonsynchronous forcing applies a much more constant forcing level over the course of one drive cycle. It can be seen throughout this chapter that for certain conditions, the nonsynchronous system is more complicated than its synchronous partner. However, for other conditions, it can also be seen that the exact

opposite is true.

6.1 Scans of Parameter a

6.1.1 Symmetric Cubic Terms

Recall the symmetric case has system parameter of $b = 10$, $c = 0.1$, $d = 10$, and $e = 0.25$. The bifurcation diagrams when a is varied negatively from 0 to -10 are shown in Figures 6.1 and 6.2. The resulting Lyapunov spectrum is shown in Figure 6.3. The early region of

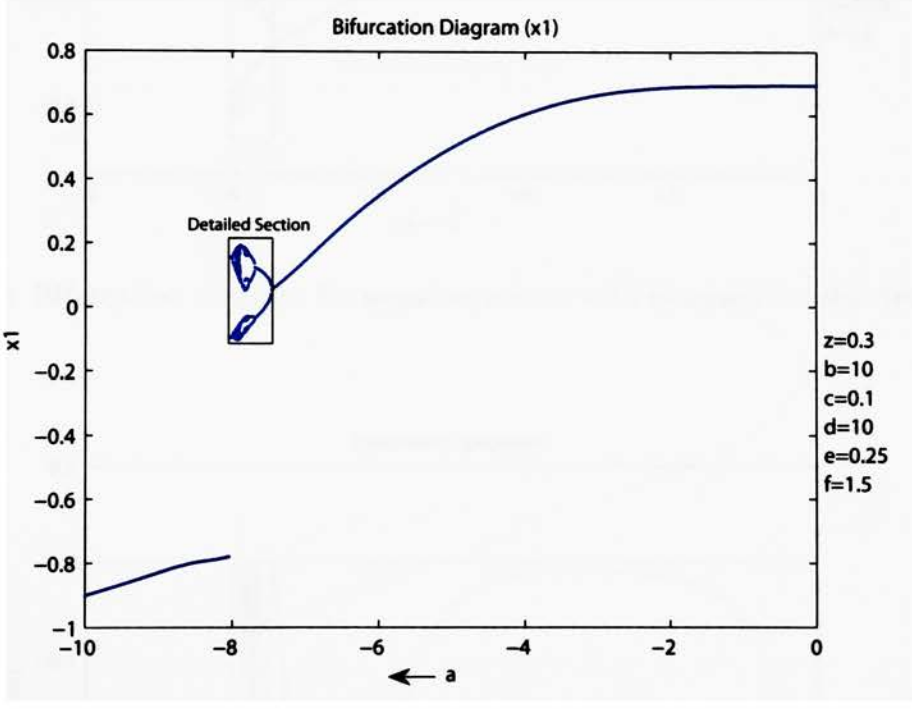


Figure 6.1: Bifurcation diagram for negative a scan with symmetric cubic terms (x_1)

chaos occurring in the synchronous system does not exist for this system. The solution remains period–1 throughout the range from 0 to -7.46. At $a = -7.47$, the system begins a period–doubling route to chaotic behavior. The period–doubling is reversed at period–8 and then continues. The chaos encountered in this region is of relatively low intensity having a maximal Lyapunov exponent of less than 0.045. The power spectra for the chaotic region have a strong period–2 harmonic content, indicating the merging of two attractors. The bifurcation diagram also shows two separate areas of high density

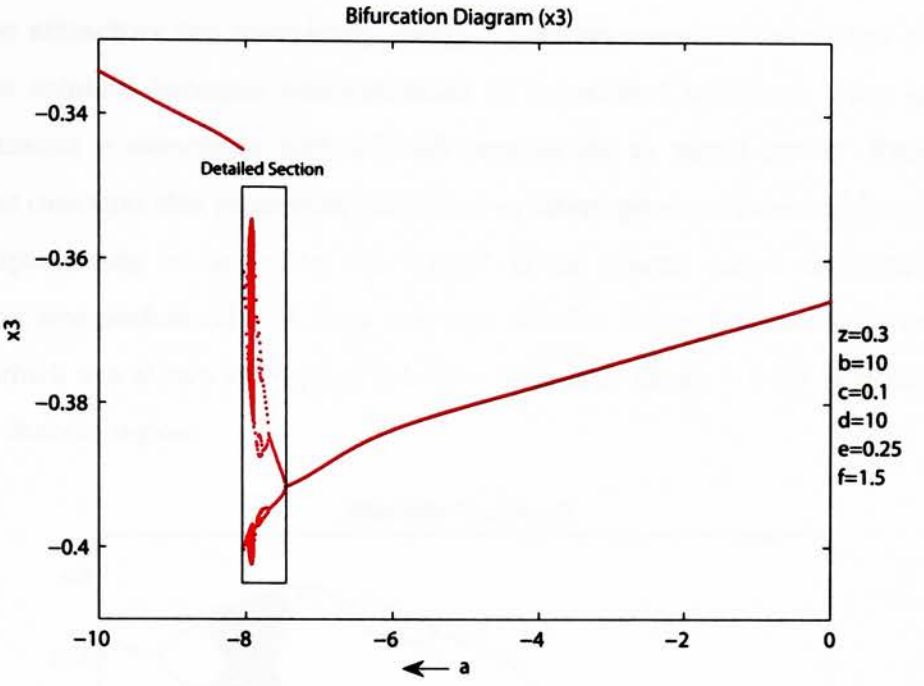


Figure 6.2: Bifurcation diagram for negative a scan with symmetric cubic terms (x_3)

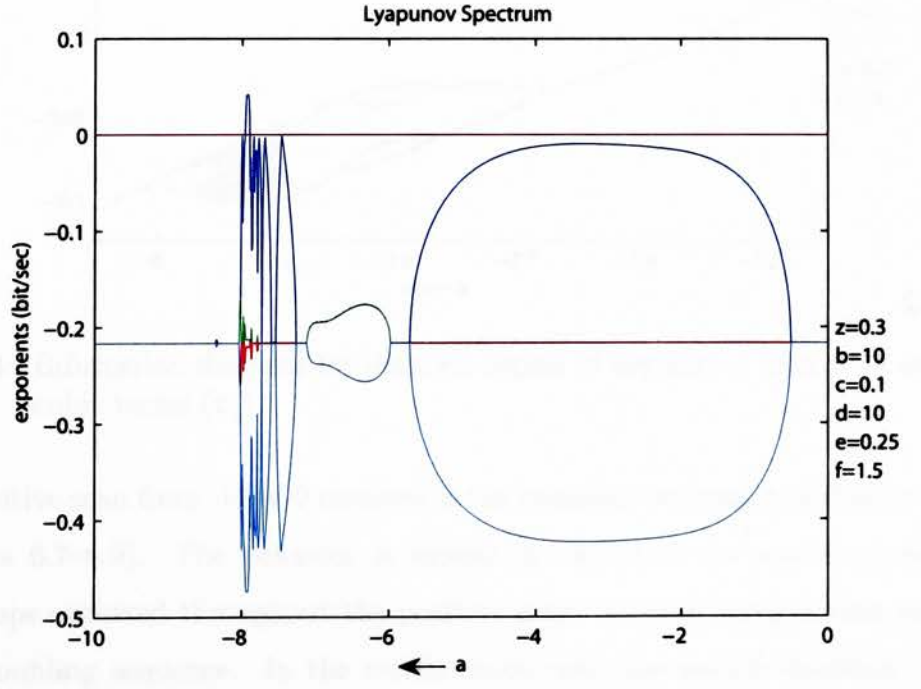


Figure 6.3: Lyapunov spectrum for negative a scan with symmetric cubic terms

Poincaré map points. The chaotic region is ended through a period–halving sequence before these attractors can completely merge. This sequence is interrupted at period–2 where the solution becomes well–entrained to a period–1 solution. Once again, the well entrainment is associated with a 20 dB drop in the x_1 signal power. Recall in the synchronous case that this parameter exhibited an interrupted period–doubling sequence. The interruption may be caused by the "ghost" of the chaotic region observed here.

This scan was performed with finer step size of 0.001 through this chaotic region, the results of which are shown in Figures 6.4–6.6. This scan shows a brief periodic window within the chaotic region.

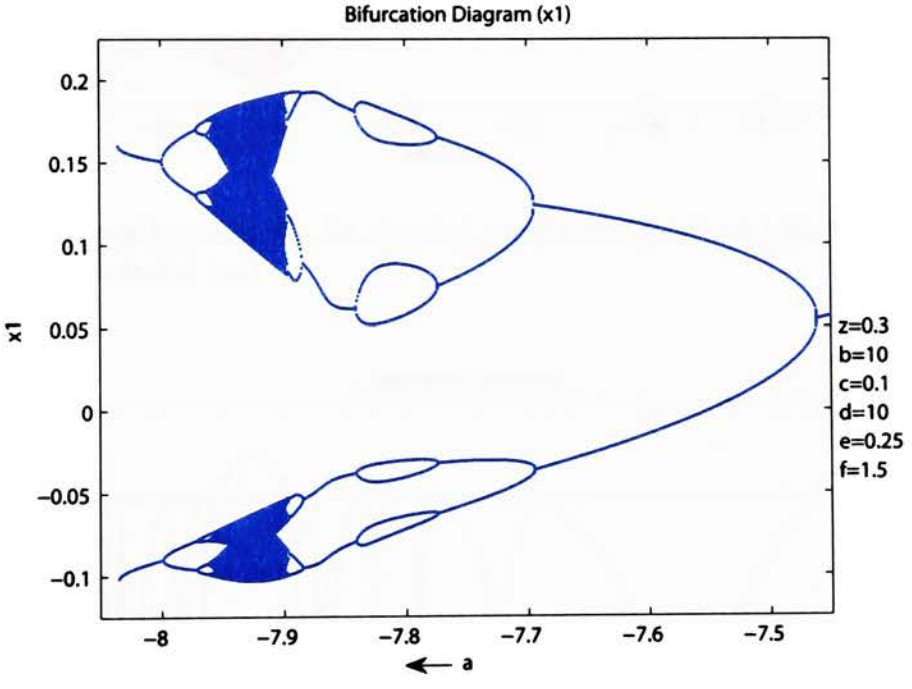


Figure 6.4: Bifurcation diagram for detailed region of negative a scan with symmetric cubic terms (x_1)

The positive scan from -10 to 0 resulted in the complete avoidance of chaotic behavior (see Figures 6.7–6.9). The behavior is similar in nature to the synchronous version. Several jumps occurred throughout the positive scan. A small jump occurs just before a period–doubling sequence. In the synchronous case, the period–doubling results in chaotic behavior. In this case, this period–doubling is interrupted by a period–1 solution occurring over both wells. The new period–1 solution undergoes another period–doubling

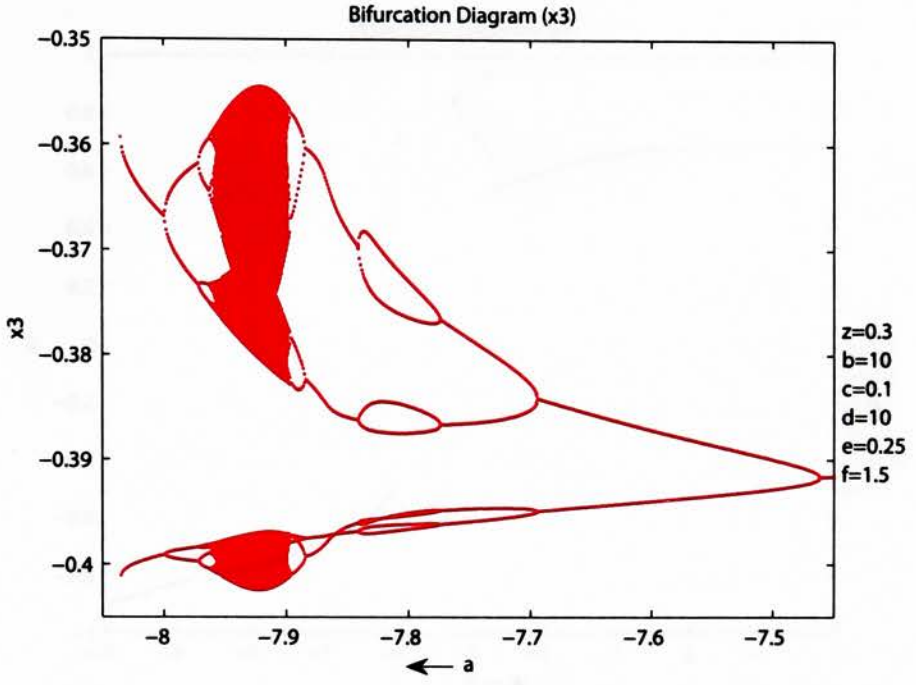


Figure 6.5: Bifurcation diagram for detailed region of negative a scan with symmetric cubic terms (x_3)

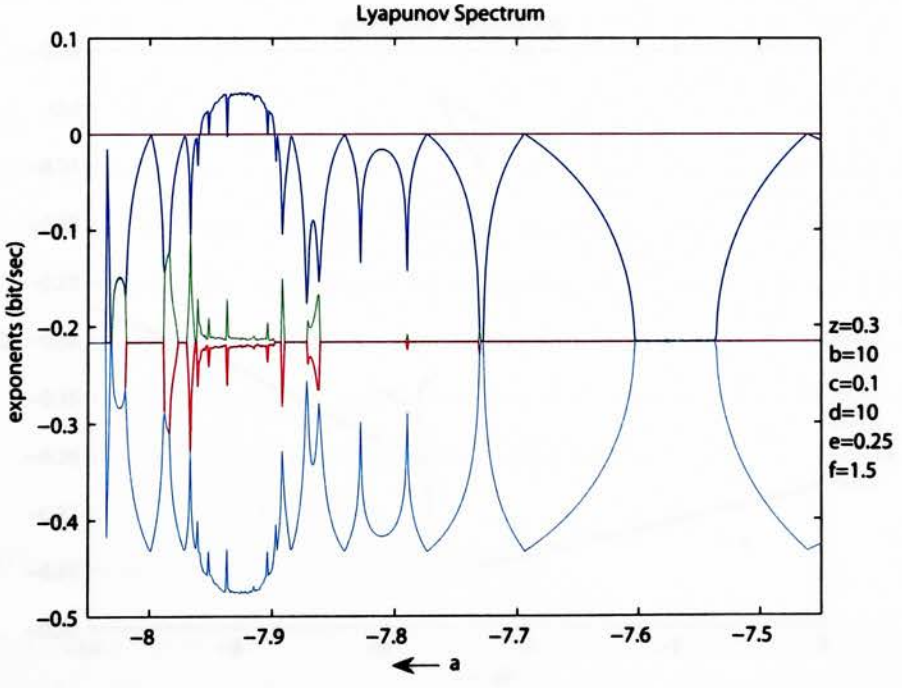


Figure 6.6: Lyapunov spectrum for detailed region of negative a scan with symmetric cubic terms

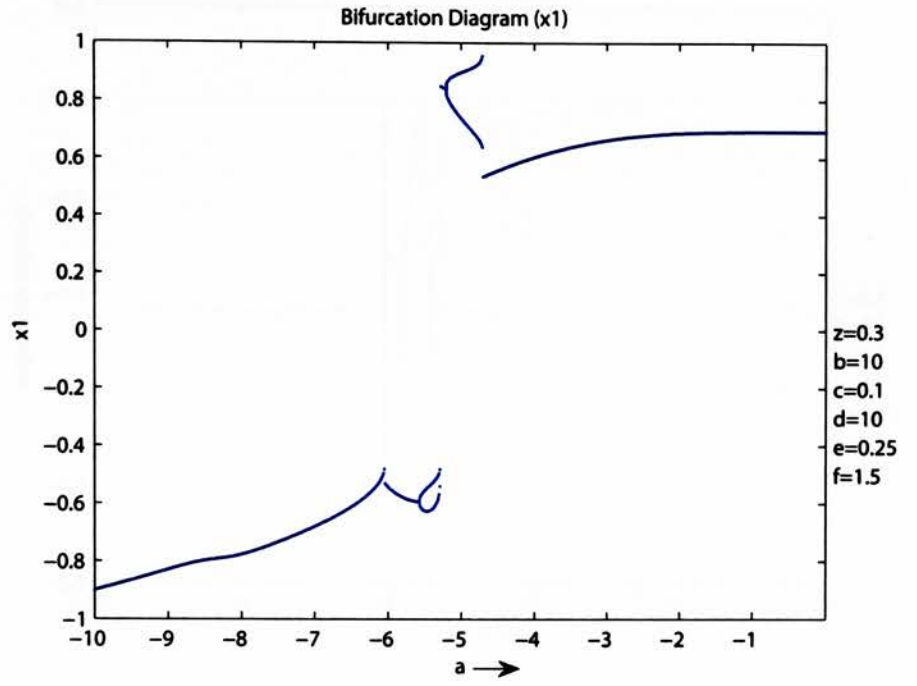


Figure 6.7: Bifurcation diagram for positive a scan with symmetric cubic terms (x_1)

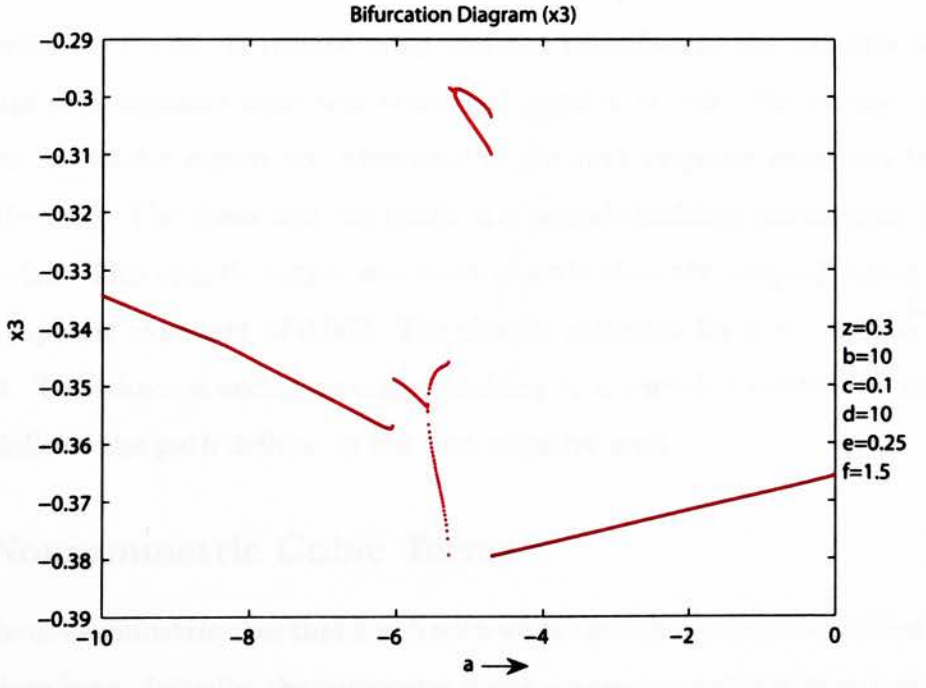


Figure 6.8: Bifurcation diagram for positive a scan with symmetric cubic terms (x_3)

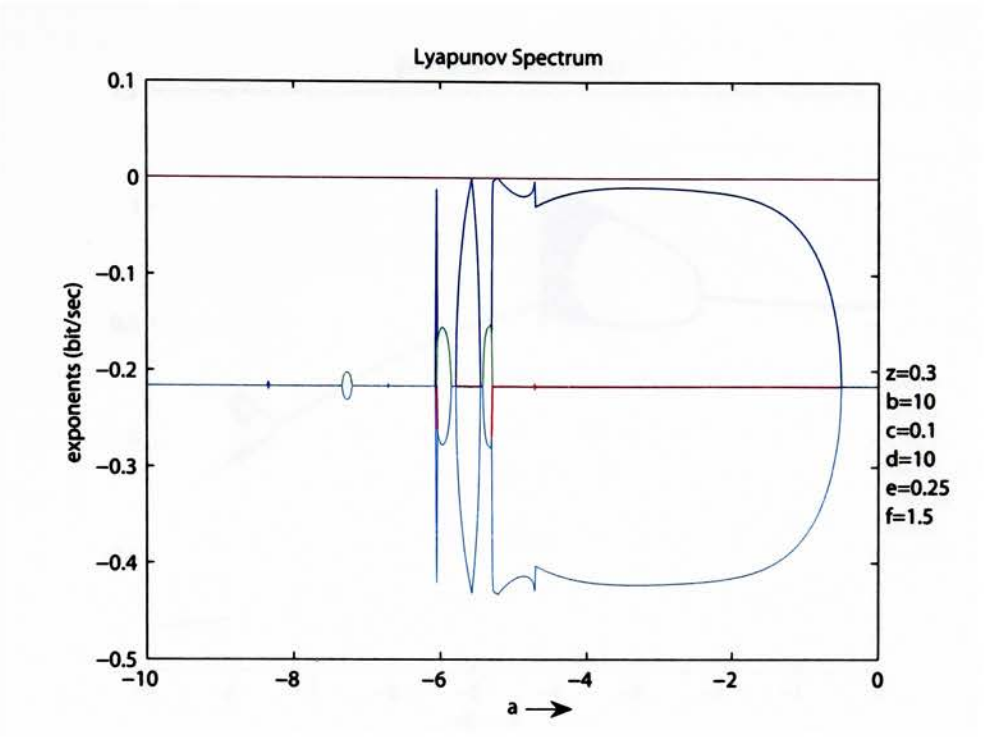


Figure 6.9: Lyapunov spectrum for positive a scan with symmetric cubic terms

that is interrupted by the more stable mirror–symmetric period–1 solution.

The positive a scan from 0 to 10 results in stable period–1 behavior with a jump occurring when $a = 0.78$. A reverse jump was not identified in the negative scan from 10 to 0, and this negative scan was continued until $a = -9$. No reverse jump was isolated, but a chaotic region not identified in the first negative scan was found (see Figures 6.10–6.12). The chaos was the result of a period–doubling phenomena beginning when $a = -2.2$. This chaotic region was more chaotic than the original region having a maximal Lyapunov exponent of 0.067. The chaotic attractor for $a = -4.16$ is shown in Figure 6.13. This chaos is ended by crisis resulting in a period–1 solution. The solution path then follows the path defined in the first negative scan.

6.1.2 Nonsymmetric Cubic Terms

Recall in the nonsymmetric case that $b = 5$ with all of the other parameter values the same as the previous scan. Initially, the parameter a was scanned negatively from 0 to -12. The solution is period–1 for values of a less than -3.05. The results from this parameter scan

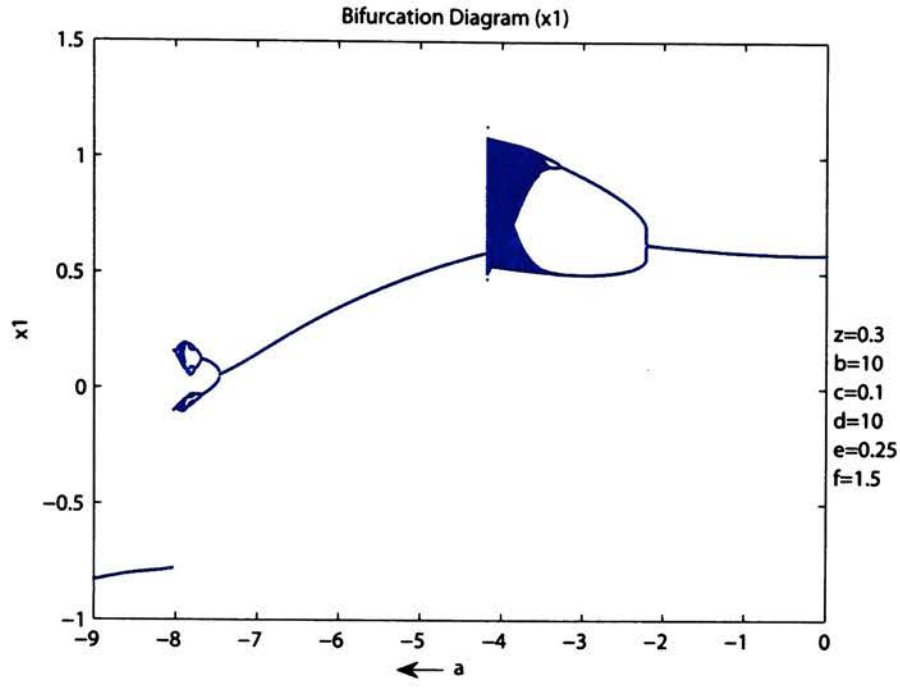


Figure 6.10: Bifurcation diagram for additional negative a scan with symmetric cubic terms (x_1)

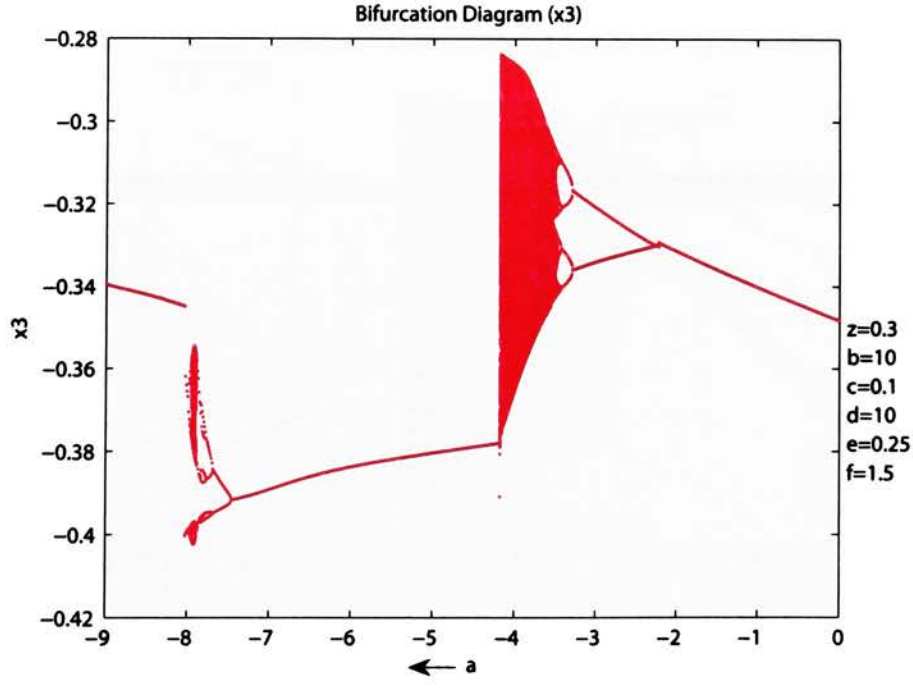


Figure 6.11: Bifurcation diagram for additional negative a scan with symmetric cubic terms (x_3)

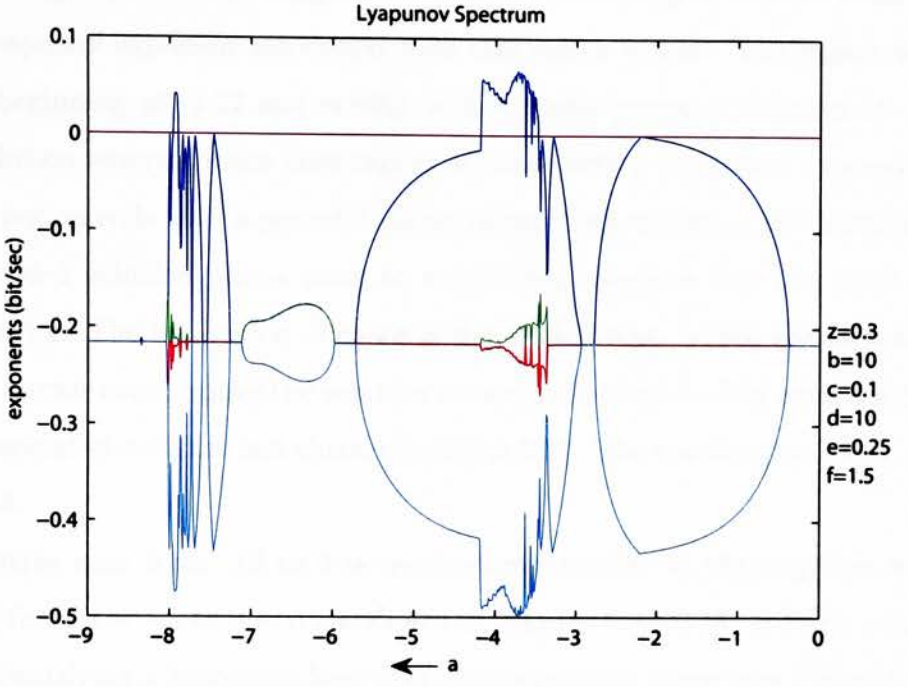
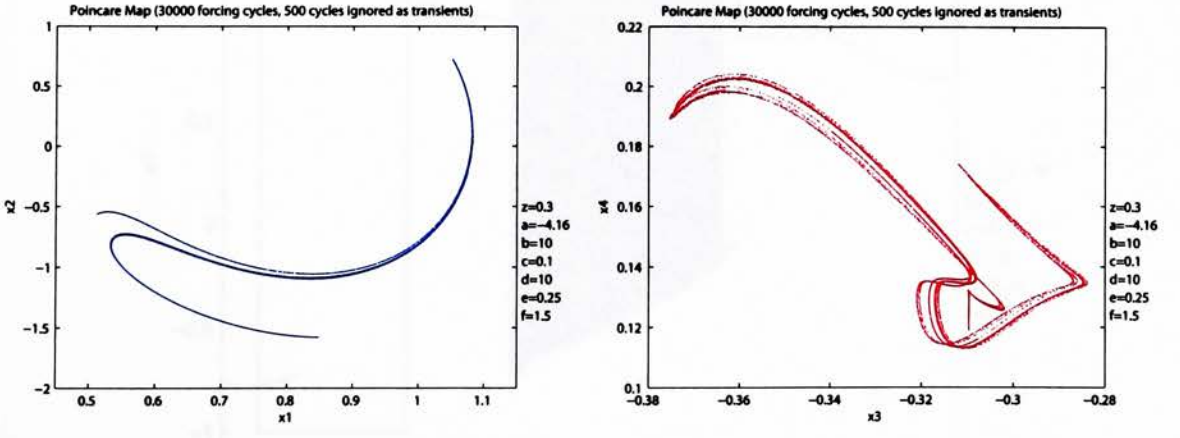


Figure 6.12: Lyapunov spectrum for additional negative a scan with symmetric cubic terms



(a) **(b)**

Figure 6.13: Chaotic Poincaré map projections for $a = -4.16$ with symmetric cubic terms corresponding to: (a) x_1–x_2 plane; (b) x_3–x_4 plane.

from $a = 0$ to $a = -5$ are shown in Figures 6.14–6.16. The solution is initially period–1 before beginning a period–doubling cascade at -1.07 resulting in chaotic behavior. The maximal Lyapunov exponent associated with this region is 0.09. The region of chaos is very brief, beginning at -1.22 and ending with a stable period–3 solution at -1.3. The period–3 solution enters a more brief region of chaos with a Lyapunov exponent of 0.14. The second region ends with a period–halving phenomena rendering the solution period–2. The period–2 solution jumps prior to a doubling sequence into the third region of chaotic behavior. The last region of chaos is the most intense region and is defined until -3.04 where a crisis event causes the solution to become period–1. The maximal Lyapunov exponent associated with the last chaotic region is 0.22. The resulting period–1 occurring at $b = -4.48$.

The positive scan from -12 to 0 is qualitatively similar to the negative scan. The scan results from $a = -5$ to $a = 0$ are shown in Figures 6.17–6.19. A jump occurs when $a = -4.16$ identifying a hysteresis loop over this parameter range (see Figure 6.20). The large chaotic region exists over the same parameter ranges as the negative scan. The

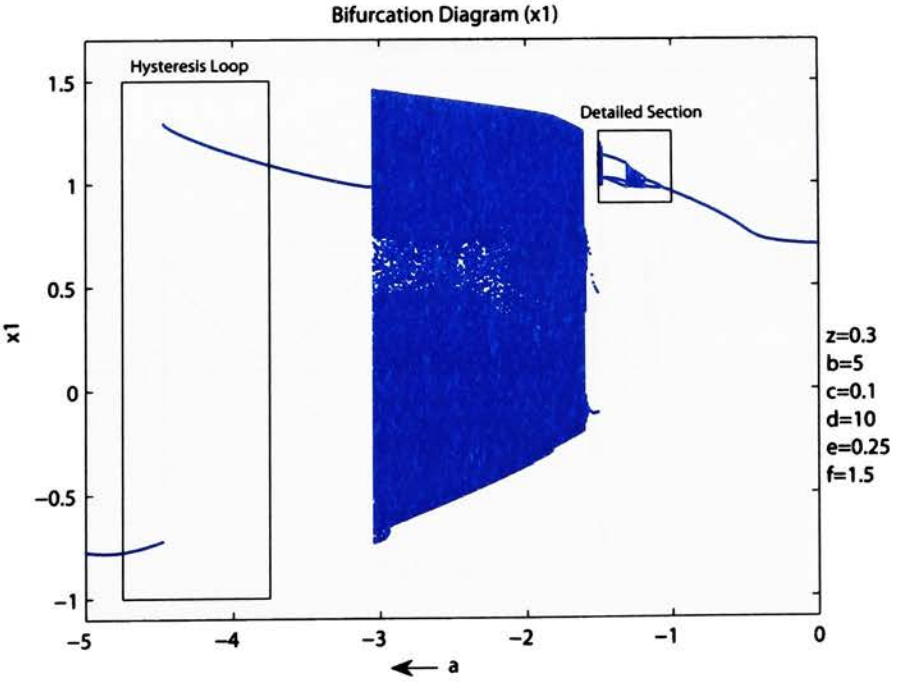


Figure 6.14: Bifurcation diagram for negative a scan with nonsymmetric cubic terms (x_1)

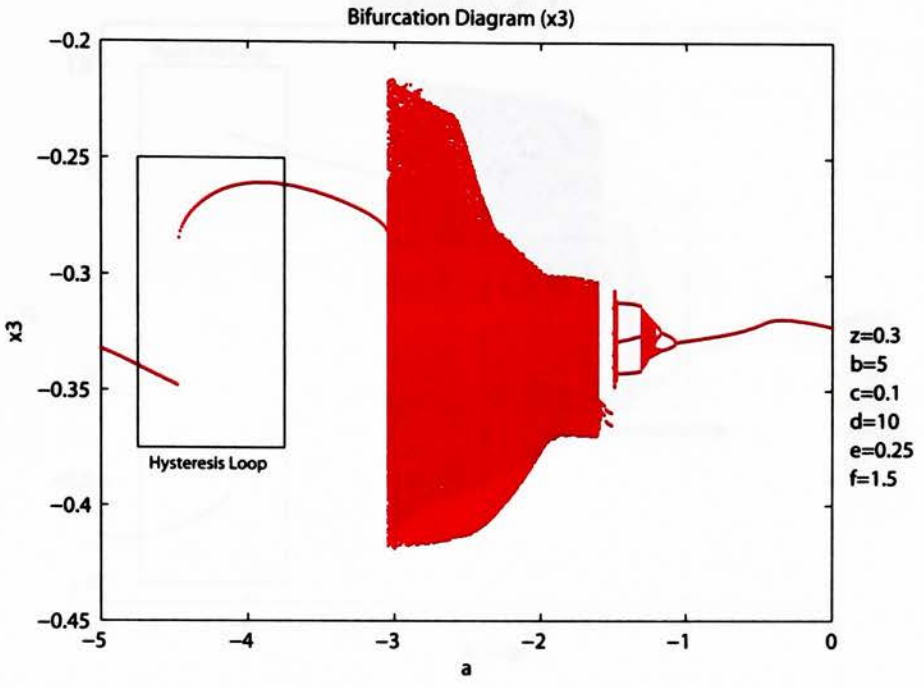


Figure 6.15: Bifurcation diagram for negative a scan with nonsymmetric cubic terms (x_3)

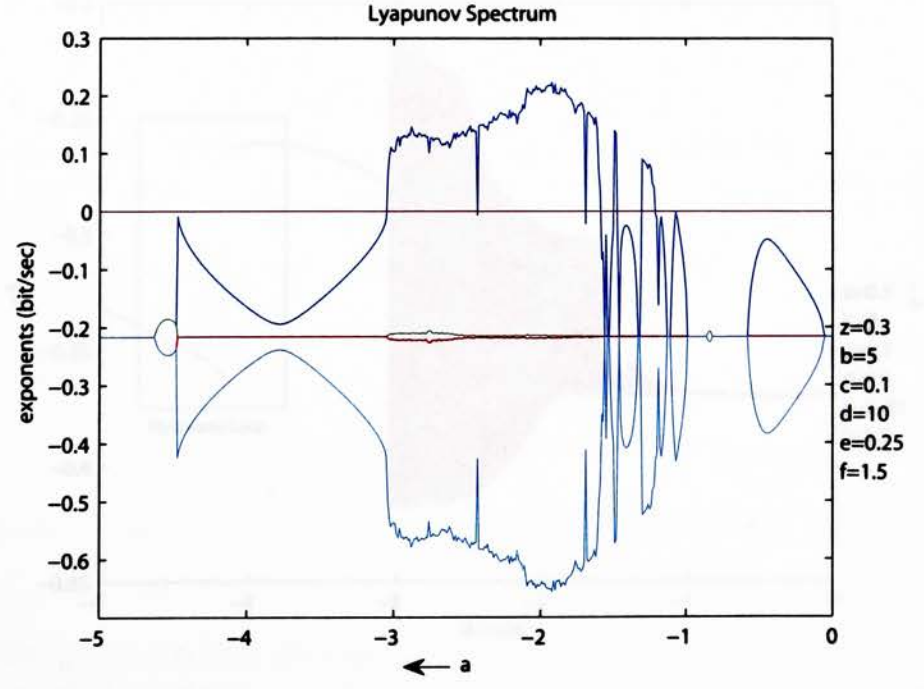


Figure 6.16: Lyapunov spectrum for negative a scan with nonsymmetric cubic terms

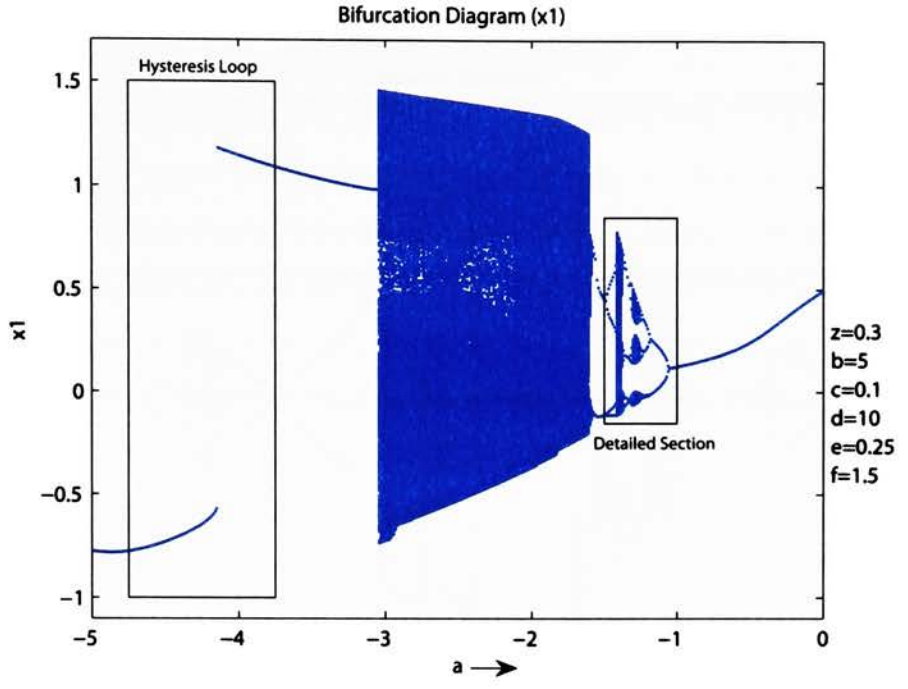


Figure 6.17: Bifurcation diagram for positive a scan with nonsymmetric cubic terms (x_1)

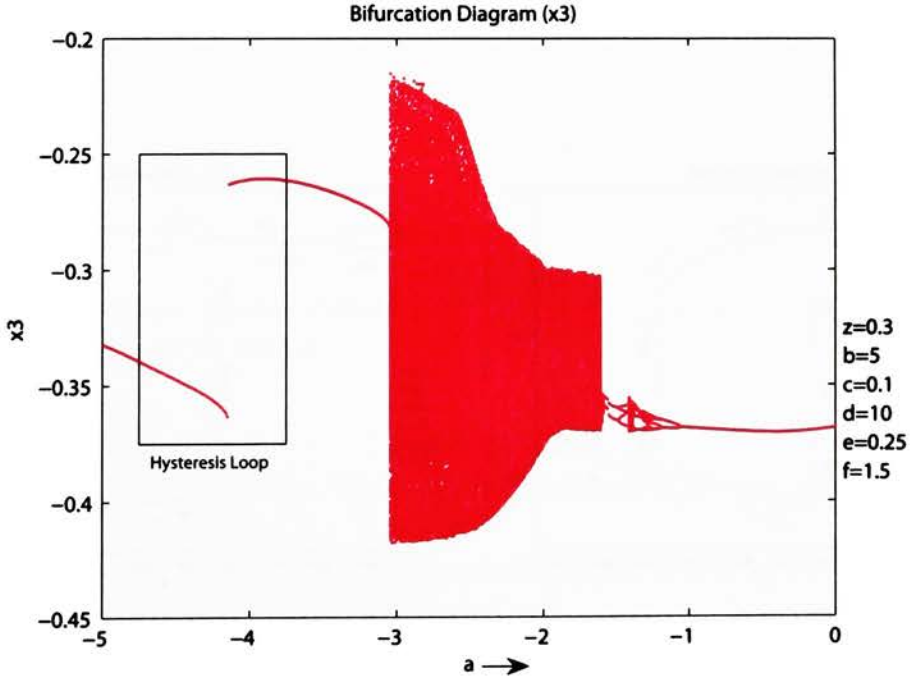


Figure 6.18: Bifurcation diagram for positive a scan with nonsymmetric cubic terms (x_3)

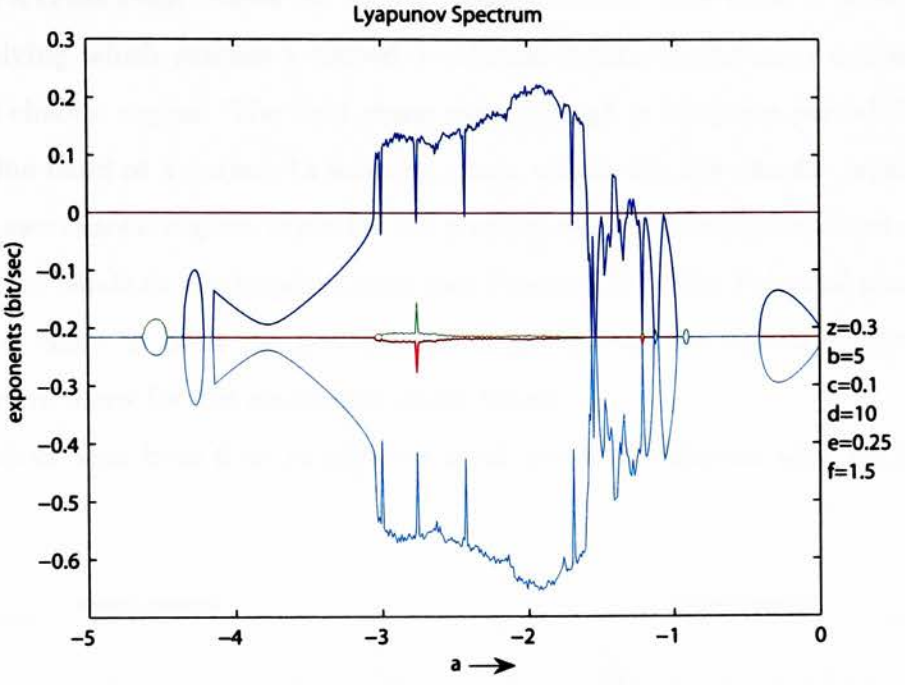


Figure 6.19: Lyapunov spectrum for negative a scan with nonsymmetric cubic terms

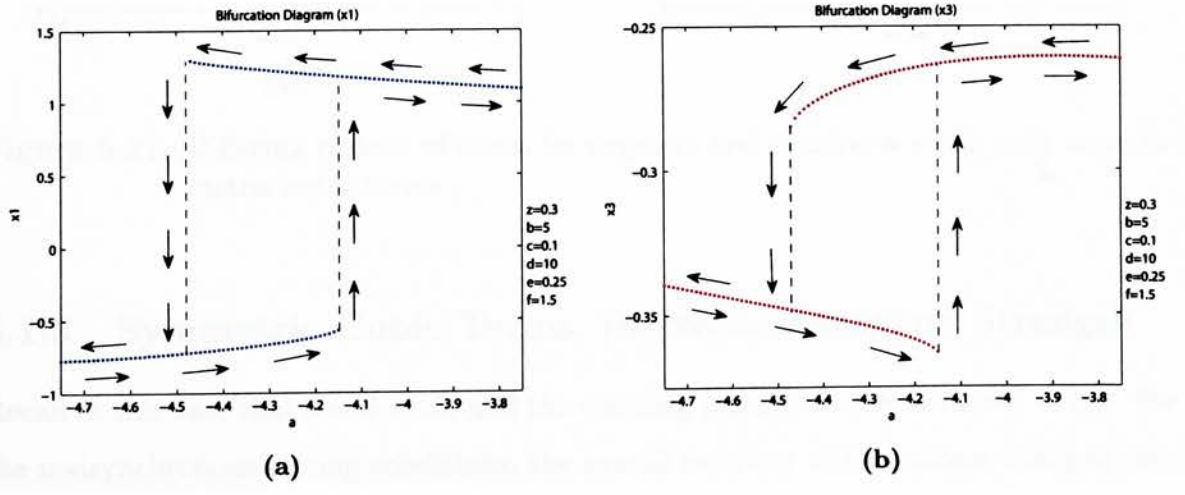


Figure 6.20: Hysteresis loop through a parameter space for a scan with nonsymmetric cubic terms

region ends through a period–halving phenomena. The period–2 solution has a small jump before a crisis event causes the second chaotic region. This band is ended through a period–halving which reaches a period–4 solution before beginning a doubling route to the third chaotic region. The final chaos ends through a complete period–halving to period–1. One band of a period–12 solution exists within the last chaotic region.

The last two chaotic regions occur for the positive scan in different locations compared to the first two bands in the negative scan (see Figure 6.21). The Poincaré maps for the large band of chaos in both the positive and negative scan are similar to those in the second negative scans for the symmetric cubic terms.

The positive scan from 0 to 10 resulted in all period–1 behavior with no jump phenomena.

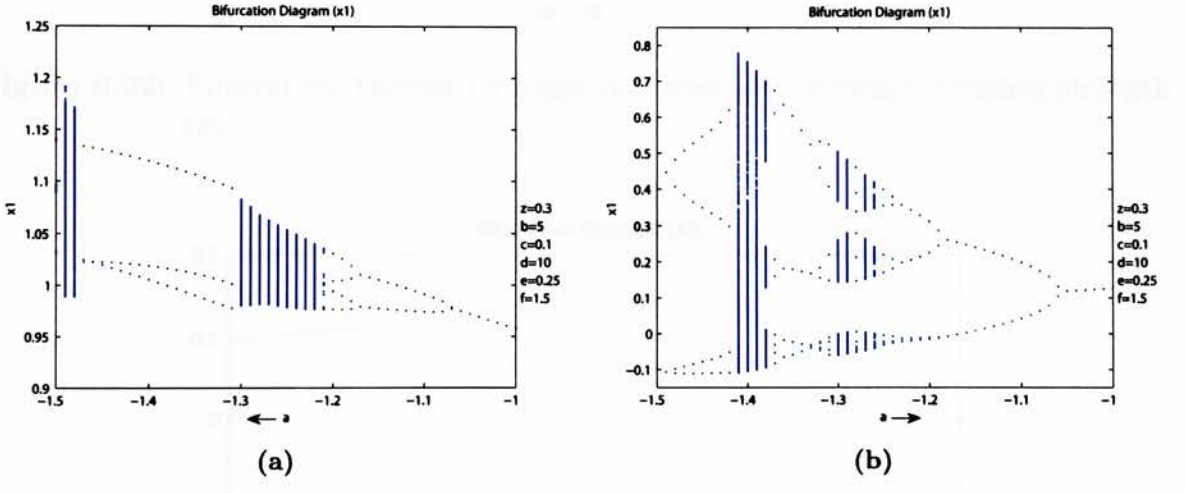


Figure 6.21: Differing regions of chaos for negative and positive a scans with nonsymmetric cubic terms

6.1.3 Symmetric Cubic Terms, Increased Coupling Strength

Recall in this case that $b = d = 10$, and the coupling parameter e is increased to 0.5. For the nonsynchronous forcing conditions, the overall behavior of the system changed very little with this increase. The behavior of this system is identical to the original symmetric case, with the exception of the completely developed chaotic region (see Figures 6.22–6.24). The solution is period–1 until undergoing a period–doubling to chaos beginning at

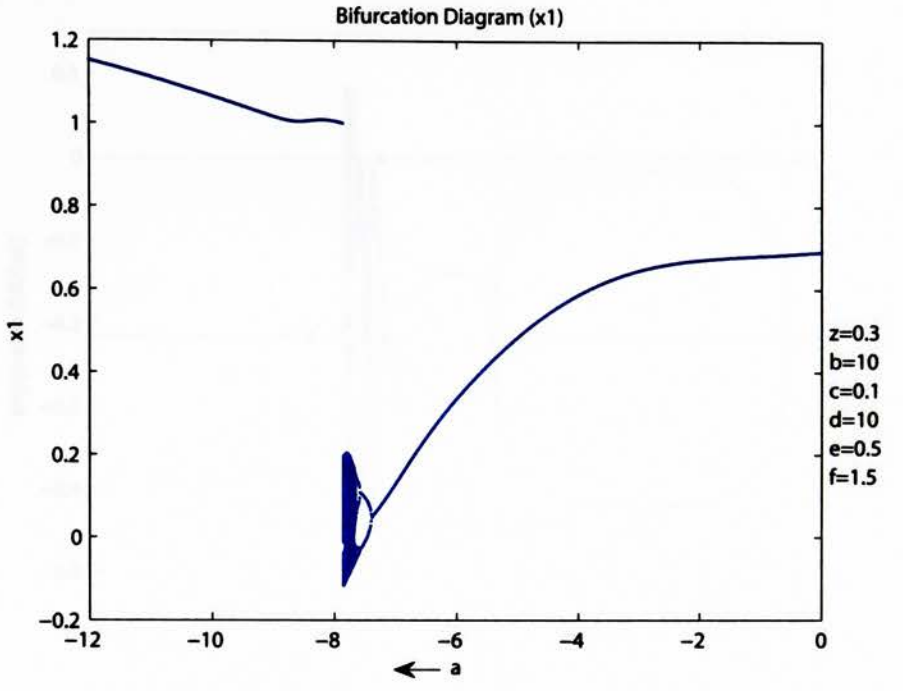


Figure 6.22: Bifurcation diagram for negative a scan with increased coupling strength (x_1)

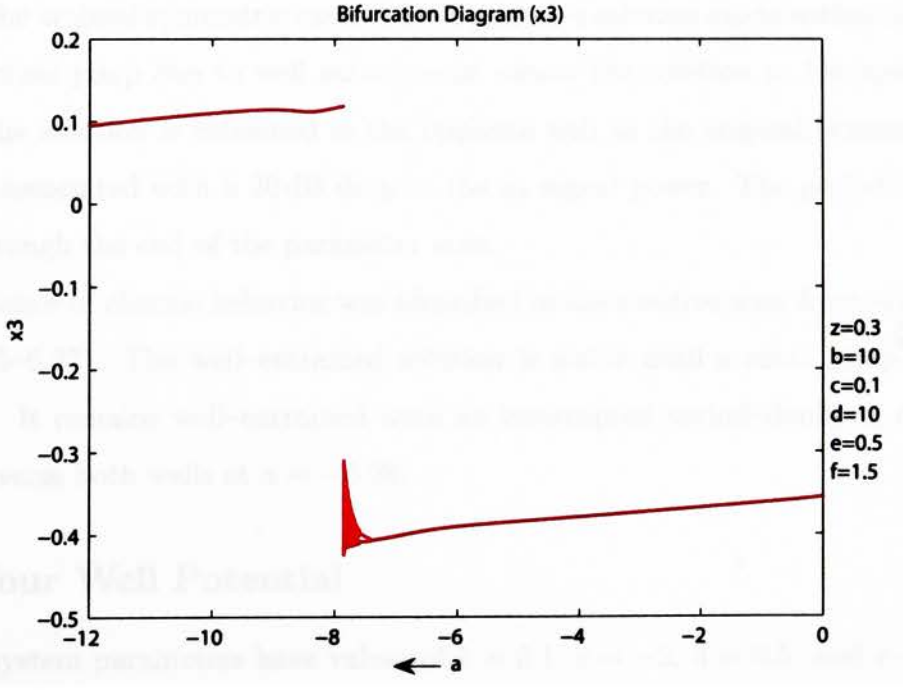


Figure 6.23: Bifurcation diagram for negative a scan with increased coupling strength (x_3)

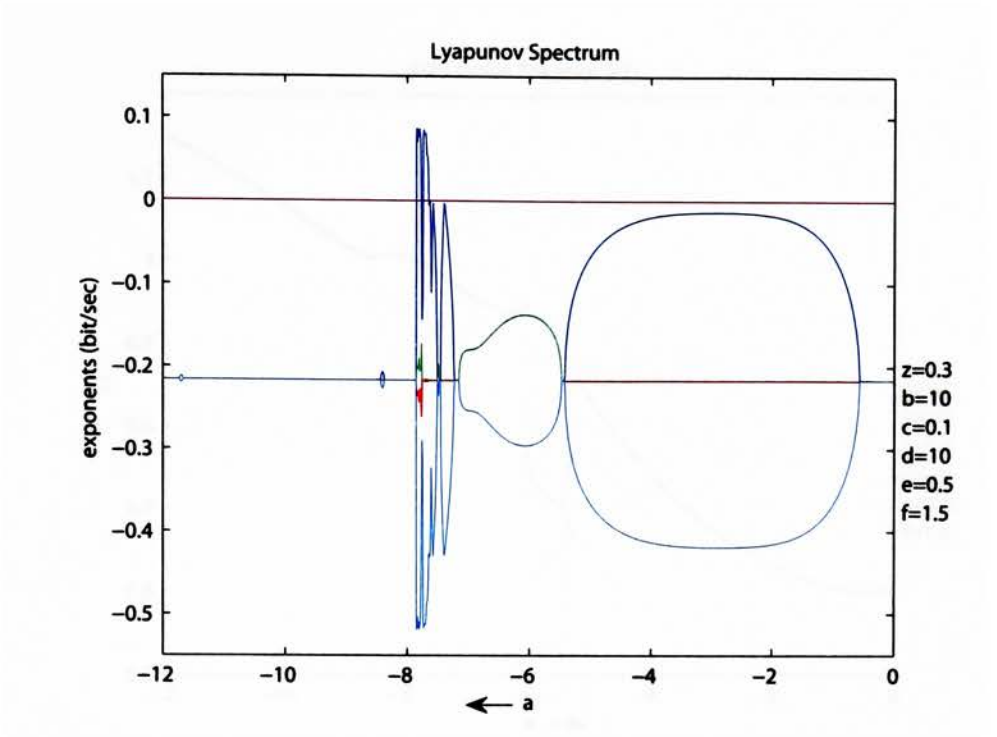


Figure 6.24: Lyapunov spectrum for negative a scan with increased coupling strength

-7.4. The band of chaos has a maximal Lyapunov exponent of 0.086, which is nearly twice as large as the original symmetric case. A brief period–4 solution exists within this chaotic region. A crisis jump due to well entrainment causes the solution to become period–1 at -7.86. The solution is entrained in the opposite well as the original symmetric scan, and is also associated with a 20 dB drop in the x_1 signal power. The period–1 solution is stable through the end of the parameter scan.

No existence of chaotic behavior was identified in the positive scan from -12 to 0 (see Figures 6.25–6.27). The well–entrained solution is stable until a small jump occurs at $a = -6.05$. It remains well–entrained until an interrupted period–doubling causes the orbit to traverse both wells at $a = -5.28$.

6.1.4 Four Well Potential

The other system parameters have values of $b = 0.1$, $c = -2$, $d = 0.5$, and $e = 0.25$ for this group of a scans. The negative scan from 0 to -10 results in all period–1 behavior. A jump, which is the result of well entrainment, takes place at $a = -1.67$. For the positive

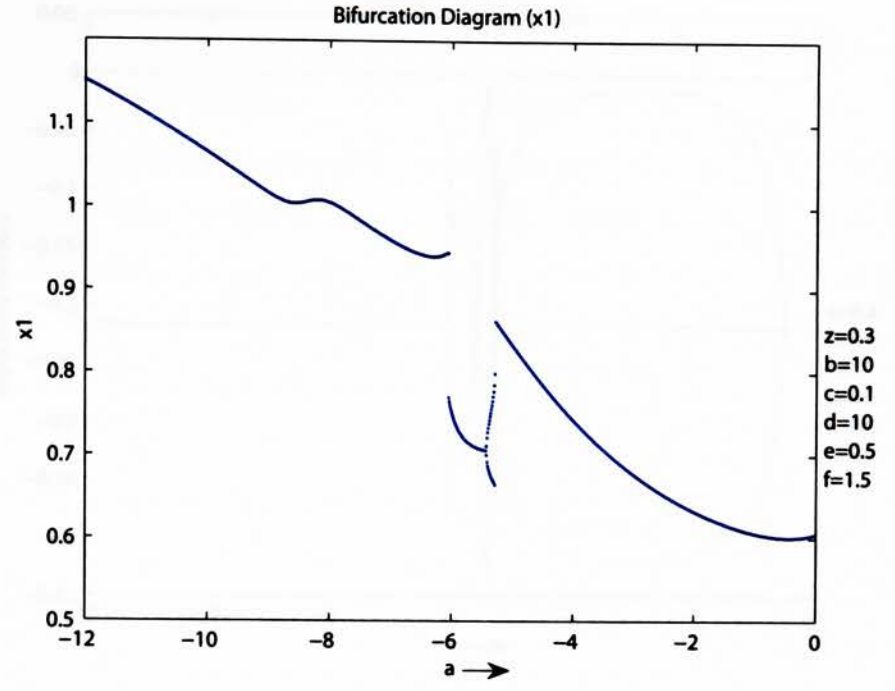


Figure 6.25: Bifurcation diagram for positive a scan with increased coupling strength (x_1)

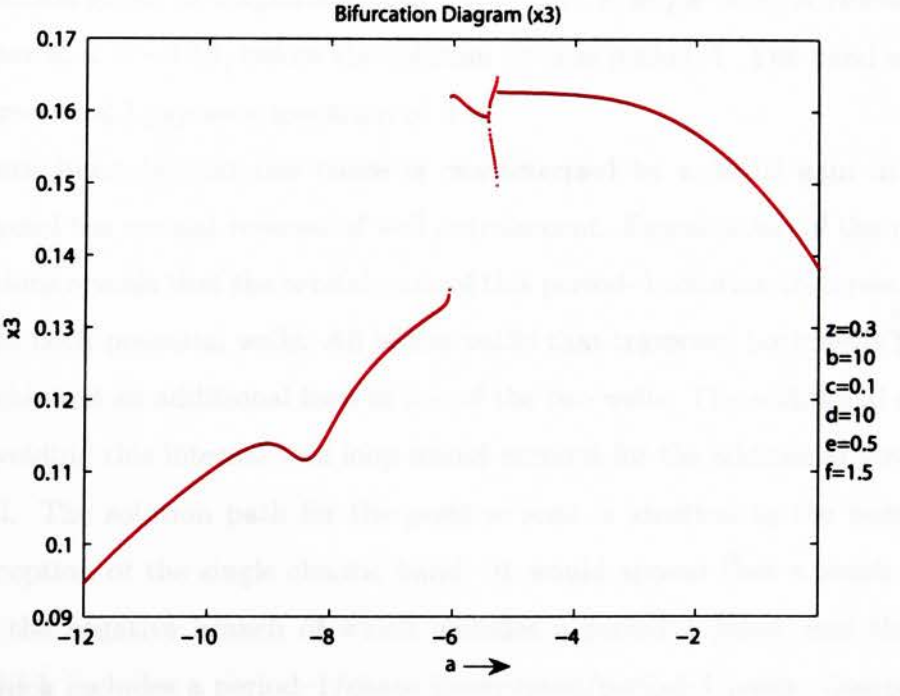


Figure 6.26: Bifurcation diagram for positive a scan with increased coupling strength (x_3)

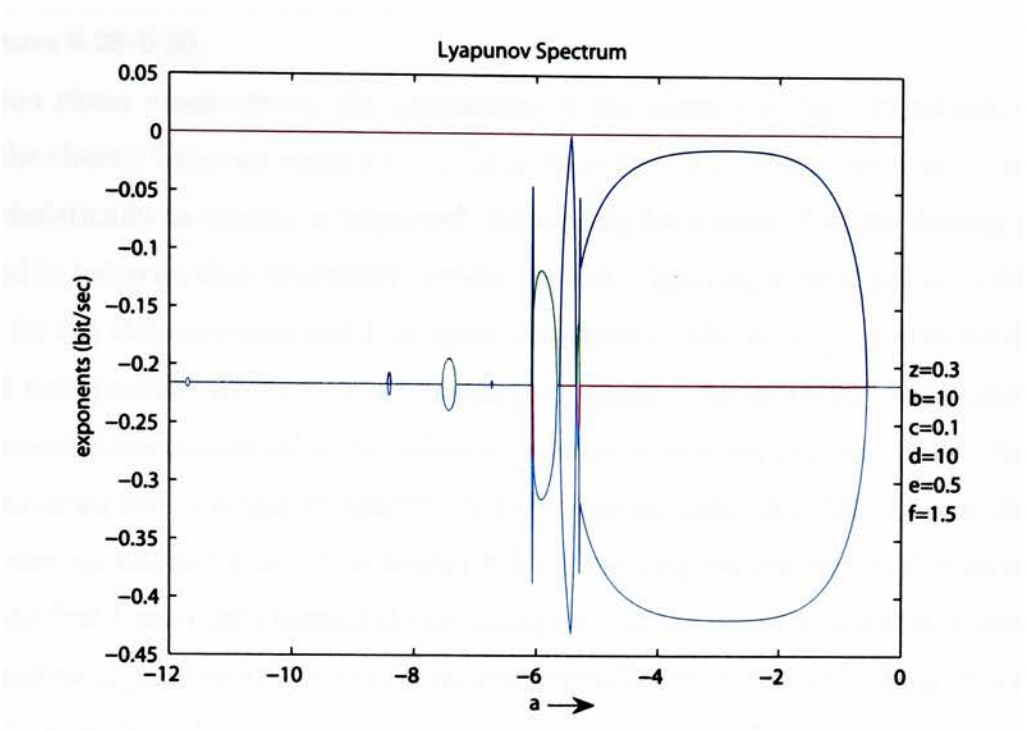


Figure 6.27: Lyapunov spectrum for positive a scan with increased coupling strength

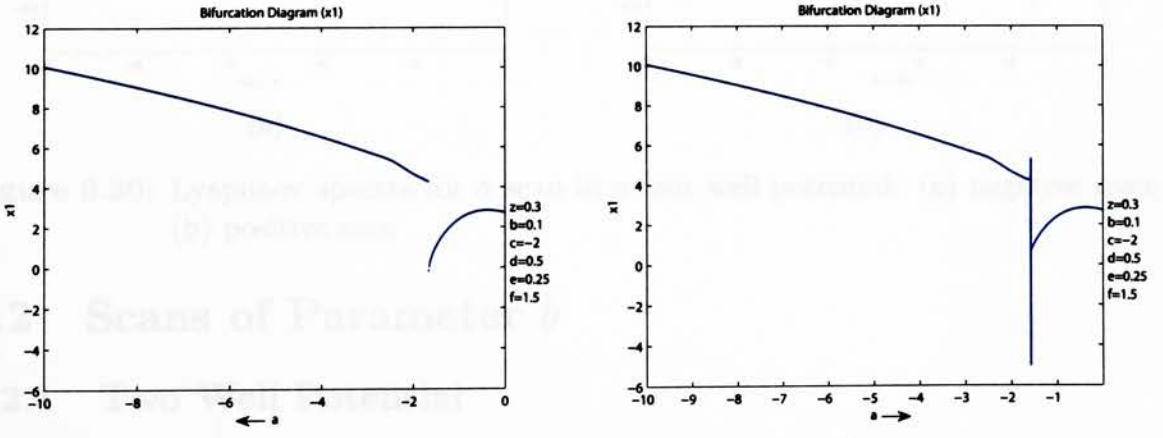
scan from -10 to 0, the solution was almost completely period–1. The well–entrained period–1 solution grows in magnitude until $a = -1.59$. A single band of chaos exists for one parameter at $a = -1.58$, before the solution returns period–1. The band of chaos at -1.58 has a maximal Lyapunov spectrum of 0.15.

The return jump beyond this chaos is characterized by a 20 dB gain in x_1 signal strength beyond the normal reversal of well entrainment. Examination of the phase portrait projections reveals that the orbital path of this period–1 solution traverses the phase space beyond both potential wells. All of the paths that traversed both wells previously examined exhibited an additional loop in one of the two wells. The additional amplitude gained by avoiding this internal well loop would account for the additional power shown in the signal. The solution path for the positive scan is identical to the negative scan with the exception of the single chaotic band. It would appear that a small hysteresis loop exists, the negative branch of which includes a period–1 jump, and the positive branch of which includes a period–1/chaos interrupted/period–1 jump. Comparisons of the negative and positive bifurcation diagrams and the Lyapunov spectrums are shown

in Figures 6.28–6.30.

Upon closer examination, the animations of the phase portrait projections did not show the chaotic behavior when $a = -1.58$ as expected. Also, the power spectra were not characteristically as chaotic as expected. Simulating for a total of 30,000 forcing periods resulted in behavior that ultimately became periodic. Ignoring approximately 4,000 drive cycles for the Poincaré map and Lyapunov exponents results in single points for the map and all non–positive values for the Lyapunov exponents. An extremely very large region of transient chaos occurs before the solution eventually becomes periodic. Since this chaos only occurred over a single parameter value, it was assumed that this lengthy transient chaos was an isolated case. The header file for the original parameter simulation only saves the first three digits beyond the decimal place for the initial conditions. Considering the sensitive dependence on initial conditions normally associated with chaos, it is entirely possible that the original scan was actually chaotic and would be chaotic regardless of the length of the simulation.

Continuing the positive scan from 0 to 10 results in period–1 orbits. The region of the parameter space contains no jumps in the solution path.



(a) (b)

Figure 6.28: Bifurcation diagrams for a scan in a four well potential (x_1): (a) negative scan; (b) positive scan

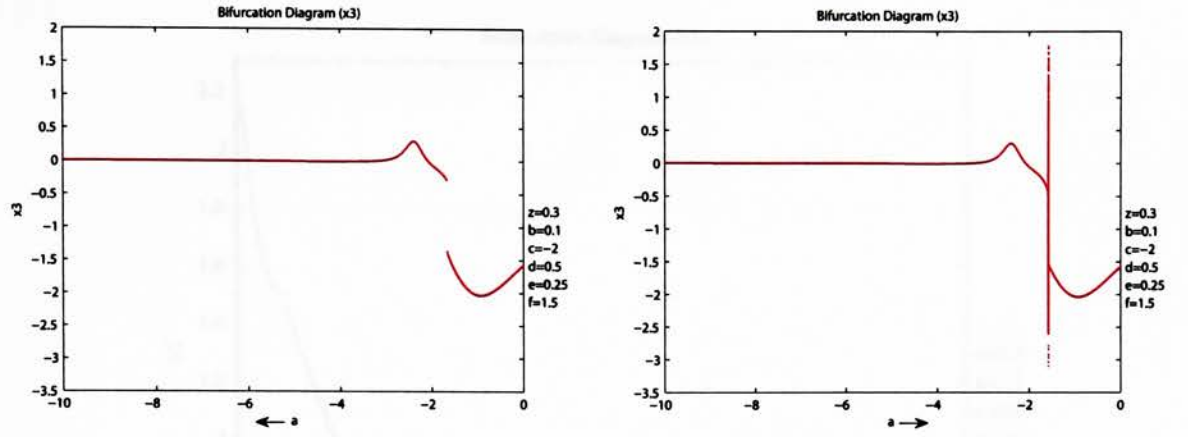


Figure 6.29: Bifurcation diagrams for a scan in a four well potential (x_3): (a) negative scan; (b) positive scan

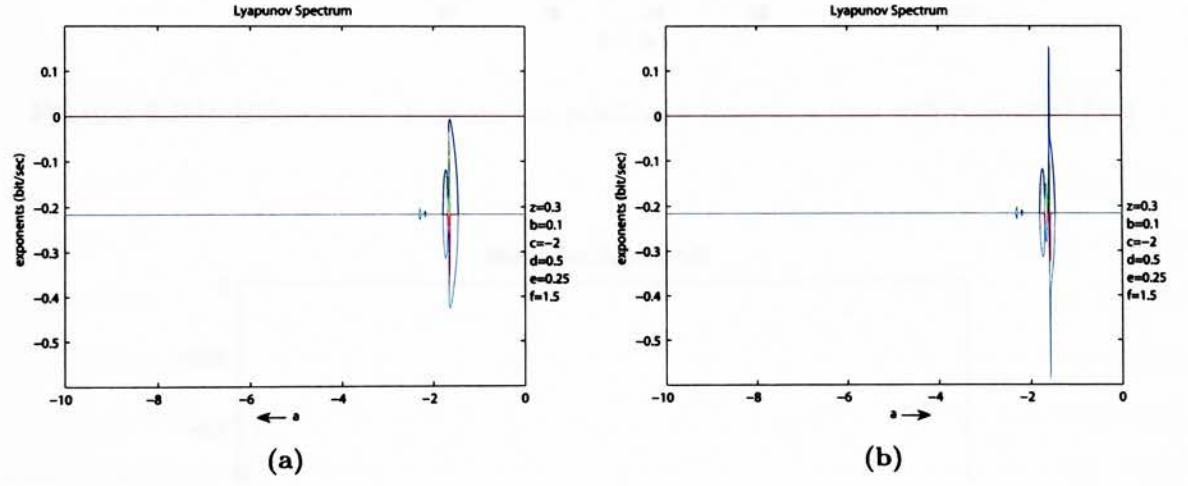


Figure 6.30: Lyapunov spectra for a scan in a four well potential: (a) negative scan; (b) positive scan

6.2 Scans of Parameter b

6.2.1 Two Well Potential

The parameter b was scanned with a positive increment from 0.01 to 35. The values of the remaining equation parameters were $a = -1$, $c = 0.1$, $d = 10$, and $e = 0.25$, which creates a two well potential symmetric about the y-axis. The results of this scan are shown in Figures 6.31–6.33. Similar to the synchronous system, a narrow region of

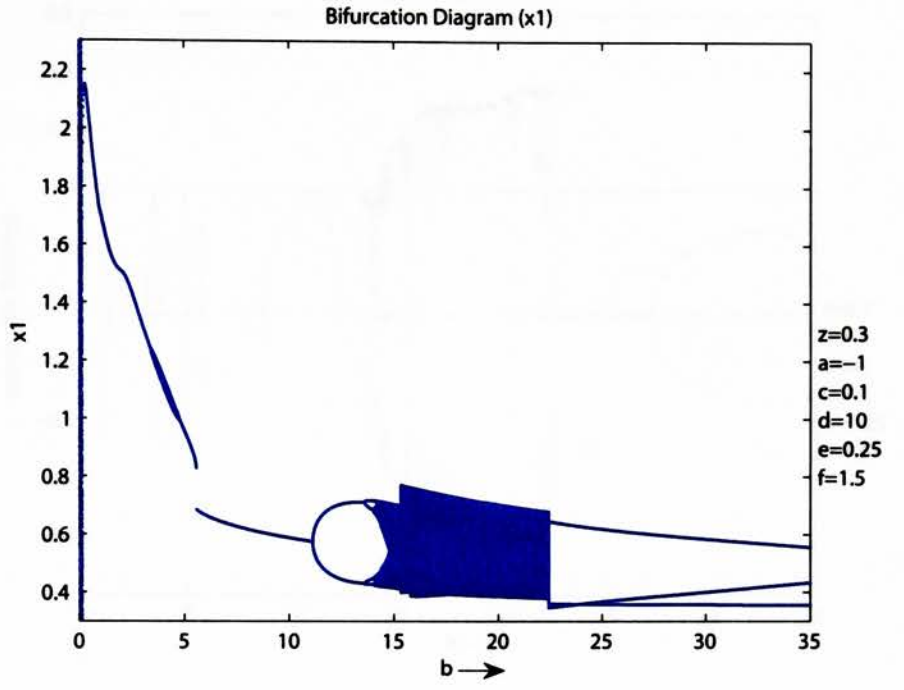


Figure 6.31: Bifurcation diagram for positive b scan in a two well potential (x_1)

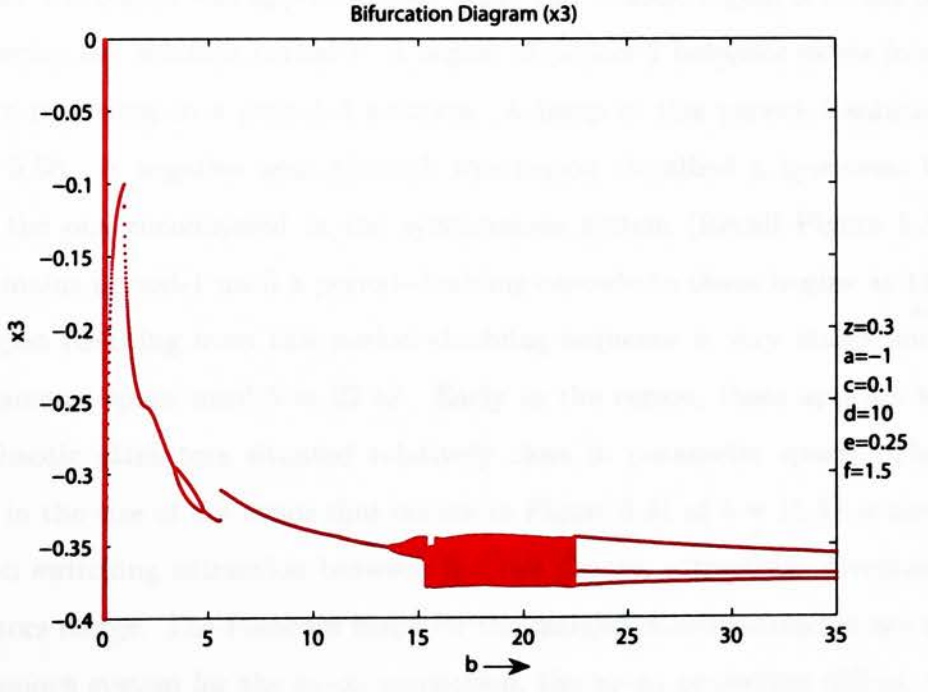


Figure 6.32: Bifurcation diagram for positive b scan in a two well potential (x_3)

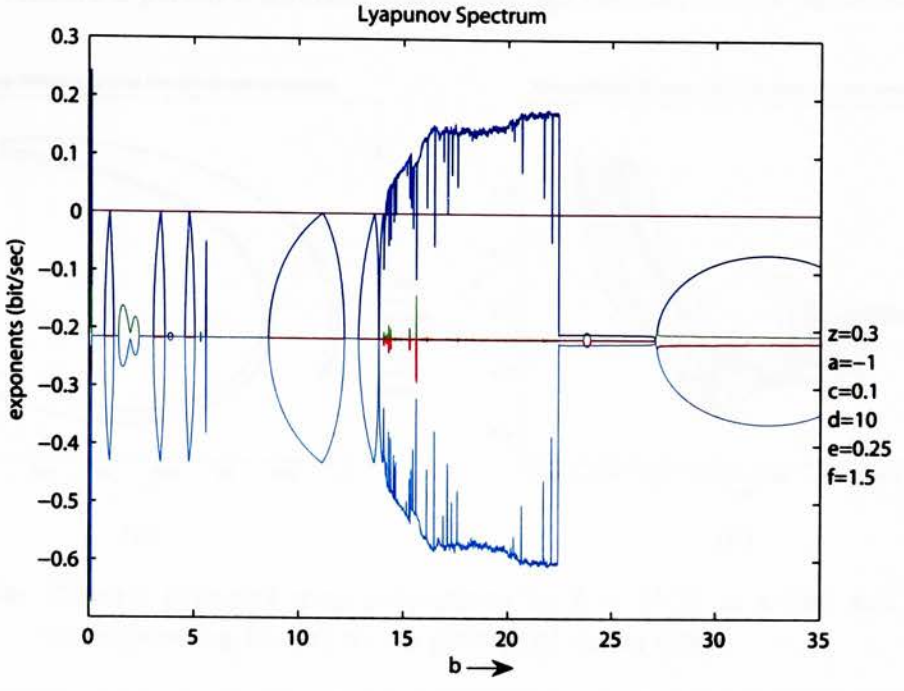
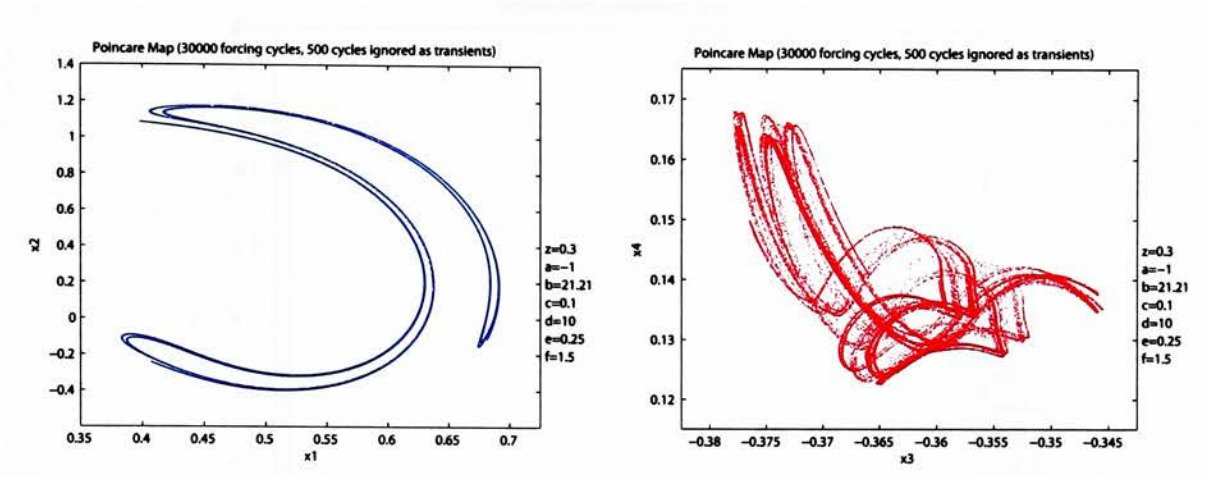


Figure 6.33: Lyapunov spectrum for positive b scan in a two well potential

high intensity chaos was found to occur for small values of b. The maximum Lyapunov exponent for the region was approximately 0.24. The chaotic region is ended by a crisis event rendering the solution period-1. A region of period–2 behavior exists from 3.41 to 4.76, before returning to a period–1 solution. A jump in this period–1 solution occurs when $b = 5.58$. A negative scan through this region identified a hysteresis loop very similar to the one encountered in the synchronous system (Recall Figure 5.33). The solution remains period-1 until a period-doubling cascade to chaos begins at 11.13. The chaotic region resulting from this period–doubling sequence is very stable and persists in the parameter space until $b = 22.43$. Early in the region, there appears to be two separate chaotic attractors situated relatively close in parameter space. The sudden "increase" in the size of the chaos that occurs in Figure 6.31 at $b = 15.34$ is the result of the solution switching attraction between the two chaotic attractors. Eventually these two attractors merge. The Poincaré maps for the merged chaotic attractor are similar to the synchronous system for the x_1–x_2 projection; the x_3–x_4 projection differs. Compare the chaotic attractors for $b = 19.6$ and $b = 21.21$ in Figures 5.32 and 6.34 respectively. A

crisis event renders a period-3 solution stable through the end of this parameter scan.



(a) (b)

Figure 6.34: Chaotic Poincaré map projections for $b = 21.21$ in a two well potential corresponding to: (a) x_1–x_2 plane; (b) x_3–x_4 plane.

6.2.2 First Four Well Potential

The parameter b was scanned from 0.01 to 16 with positive increments. The remaining system parameters were fixed at $a = -1$, $c = -5$, $d = 0.5$, and $e = 0.25$. Chaos exists for three values of b early in the parameter space. The remaining solution path is period–1. The results of this scan for b from 0.01 to 1 are shown in Figures 6.35–6.37. The chaos occurs about four wells. A solution entrainment jump occurs when $b = 0.4$, and the solution becomes entrained in one of the y-axis wells. A corresponding 15 dB drop occurs in the x_3 signal at this point.

6.2.3 Second Four Well Potential

The parameter b was varied from 0.01 to 12 for the second four well potential scan. The remaining equation parameters have values of $a = -4.5$, $c = -3.5$, $d = 0.5$, and $e = 0.25$. This system exhibited period–1 behavior over the entire parameter space considered. One jump resulting from well entrainment occurred when $b = 0.77$. The jump resulted in a drop in power of approximately 4 dB in the x_1 signal and approximately 10 dB in the x_3 signal.

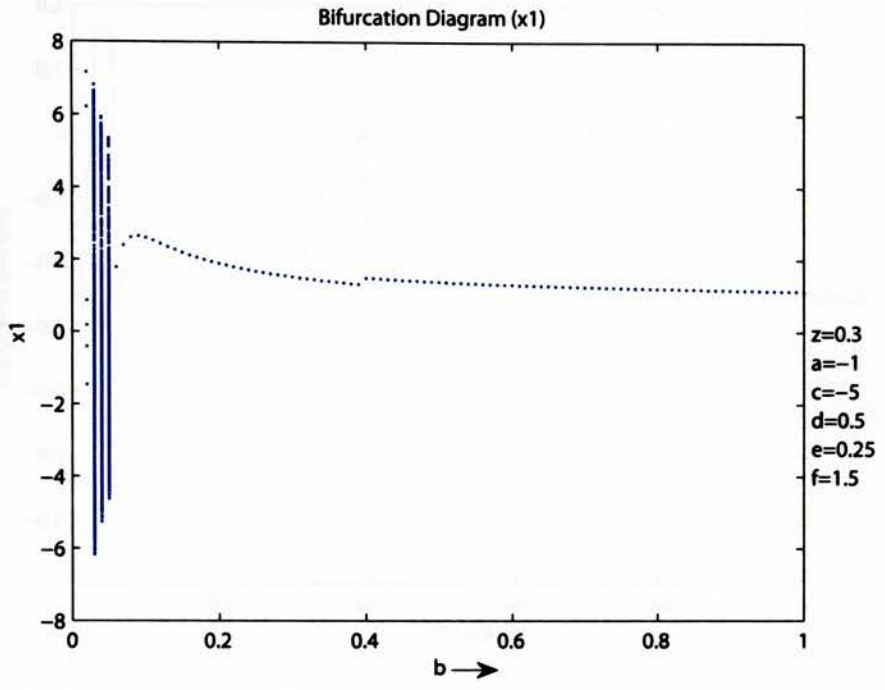


Figure 6.35: Bifurcation diagram for positive b scan in first four well potential (x_1)

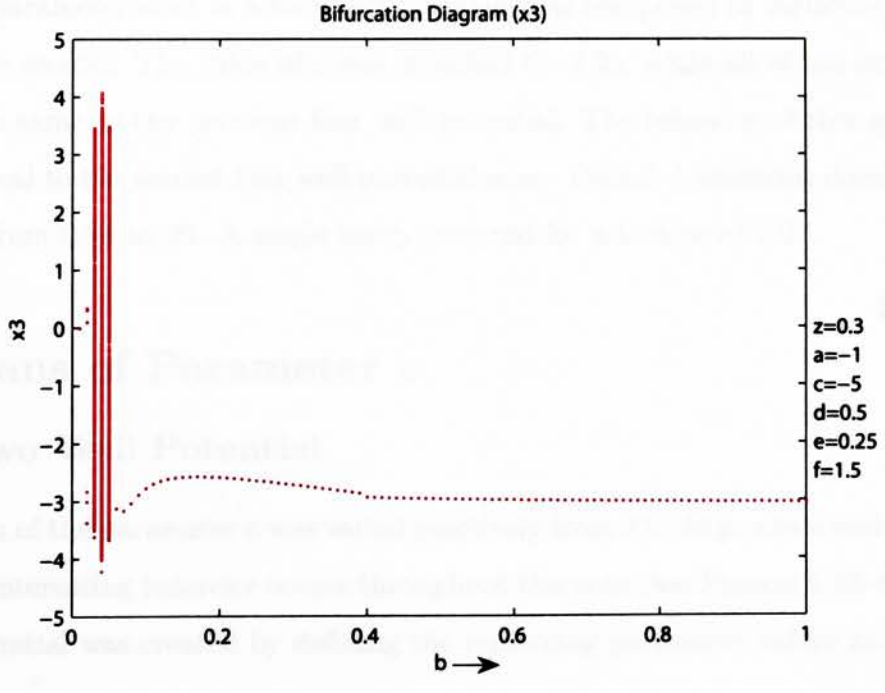


Figure 6.36: Bifurcation diagram for positive b scan in first four well potential (x_3)

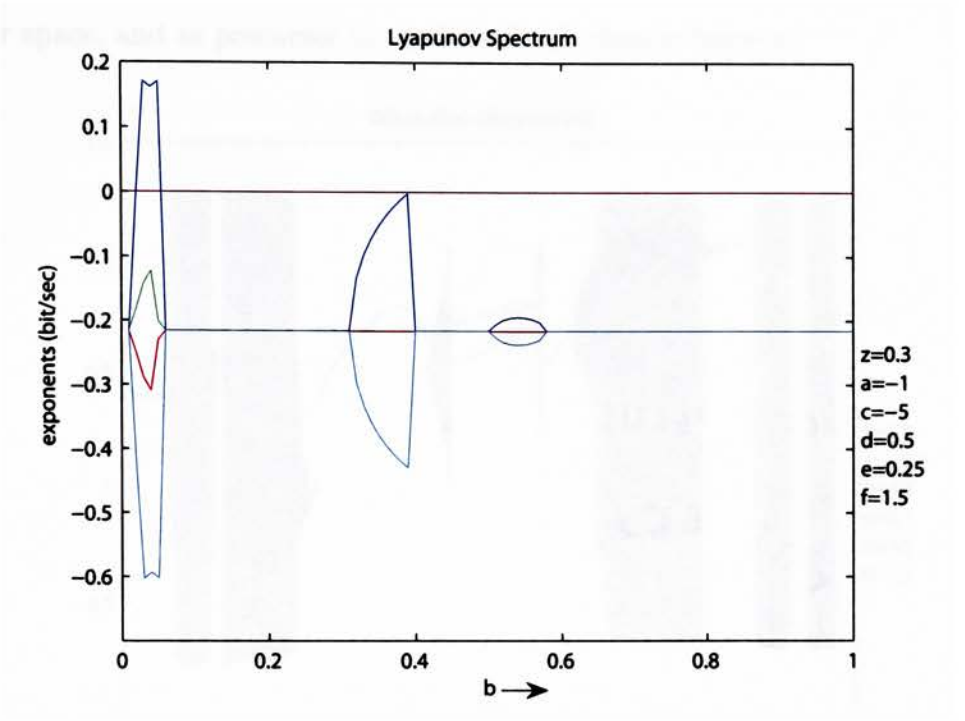


Figure 6.37: Lyapunov spectrum for positive b scan in first four well potential

6.2.4 Third Four Well Potential

The third b parameter scan in a four well potential was comprised of variables that were similar to the second. The value of c was modified to -2.25, while all of the other values remained the same as the previous four well potential. The behavior of this system was nearly identical to the second four well potential scan. Period–1 solutions dominated the entire span from 0.01 to 20. A single jump occurred for a b value of 0.97.

6.3 Scans of Parameter e

6.3.1 Two Well Potential

The first scan of the parameter e was varied positively from 0 to 40 in a two well potential. A variety of interesting behavior occurs throughout this scan (see Figures 6.38–6.40). The two well potential was created by defining the remaining parameter values as $a = -5.5$, $b = 10$, $c = 0.1$, and $d = 10$. Similar to the synchronous system, this parameter scan resulted in the emergence of quasiperiodicity existing for extended regions of the

parameter space, and as precursor to, and result of, chaotic behavior.

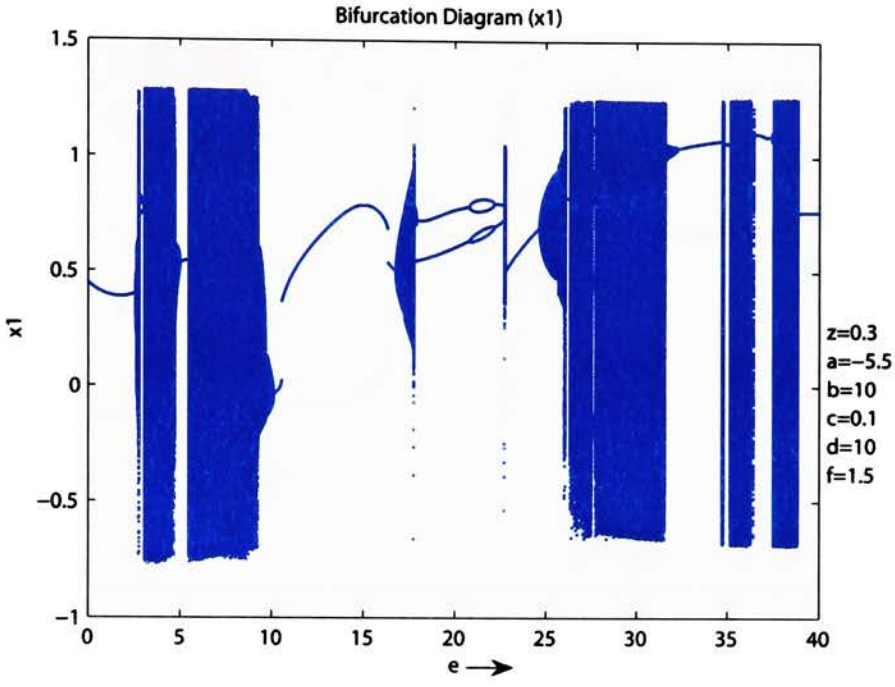


Figure 6.38: Bifurcation diagram for positive e scan in a two well potential (x_1)

The solution is initially period–1. A small quasiperiodic region exists before the onset of the first chaotic region. The chaos is ended by well entrainment that results in period–1 behavior. A sudden emergence of a period–16 solution occurs for $e = 2.84$. This solution undergoes a period–halving cascade, and returns to a period–1 orbit. Crisis causes the solution enter the second chaotic region at $e = 3.07$. The chaotic region is ended by quasiperiodicity that occurs for $e = 4.73$. The quasiperiodicity culminates in period–1 behavior at 5.02.

Crisis causes the system to enter the third region of chaos. The second Lyapunov exponent oscillates around zero for most of this region. Quasiperiodicity, culminating in a period–1 orbit, ends this chaotic band. The quasiperiodic attractor for $e = 10.05$ is shown in Figure 6.41.

Jumps occur in this period–1 solution for $e = 10.56$ and $e = 16.33$. Quasiperiodicity flanks both sides of a brief region of low intensity chaos that occurs from $e = 17.23$ to $e = 17.76$. The region also contains the near zero and slightly positive second Lyapunov

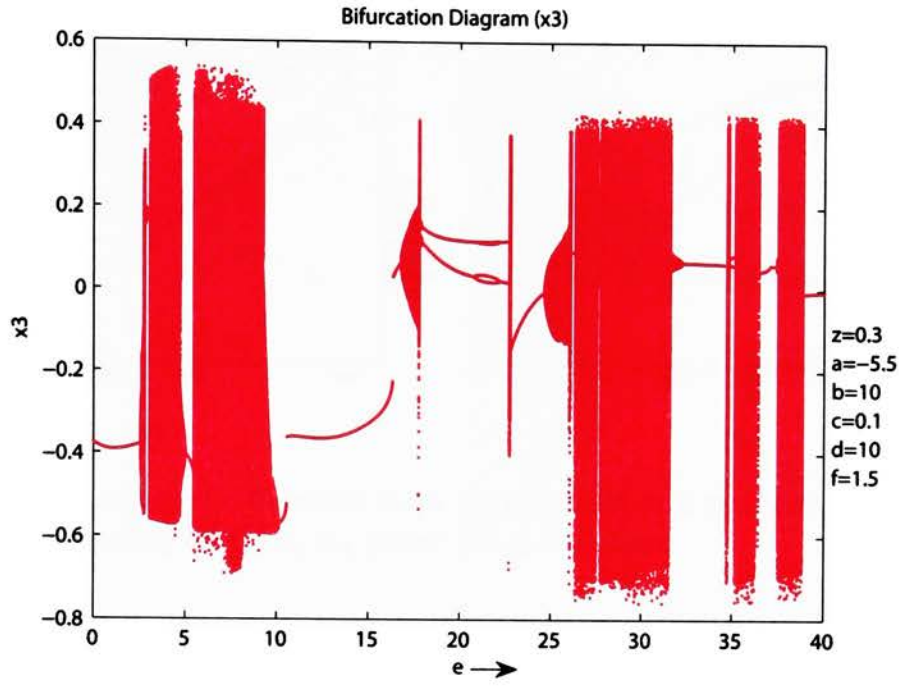


Figure 6.39: Bifurcation diagram for positive e scan in a two well potential (x_3)

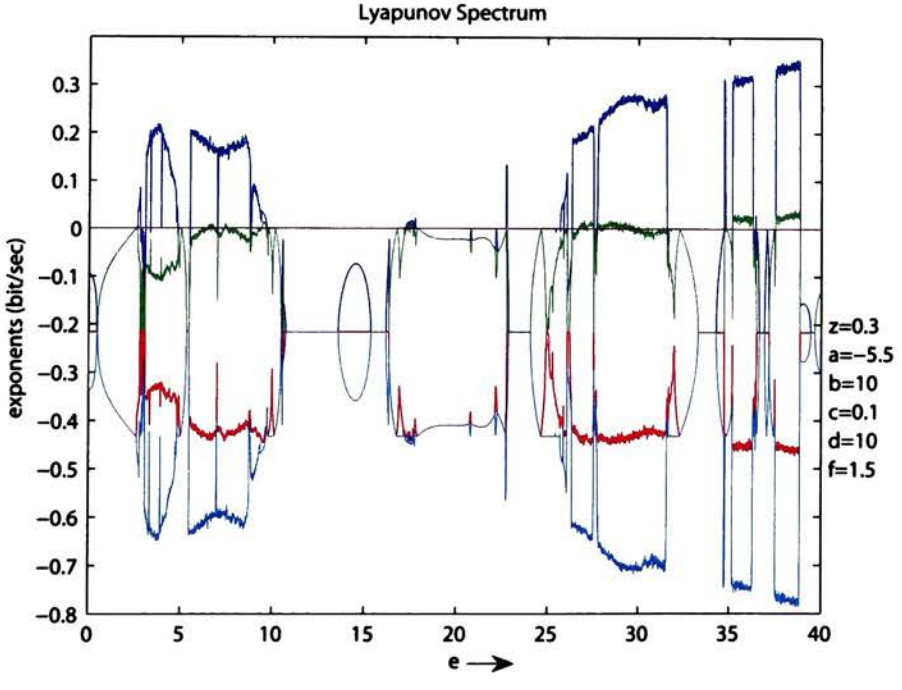
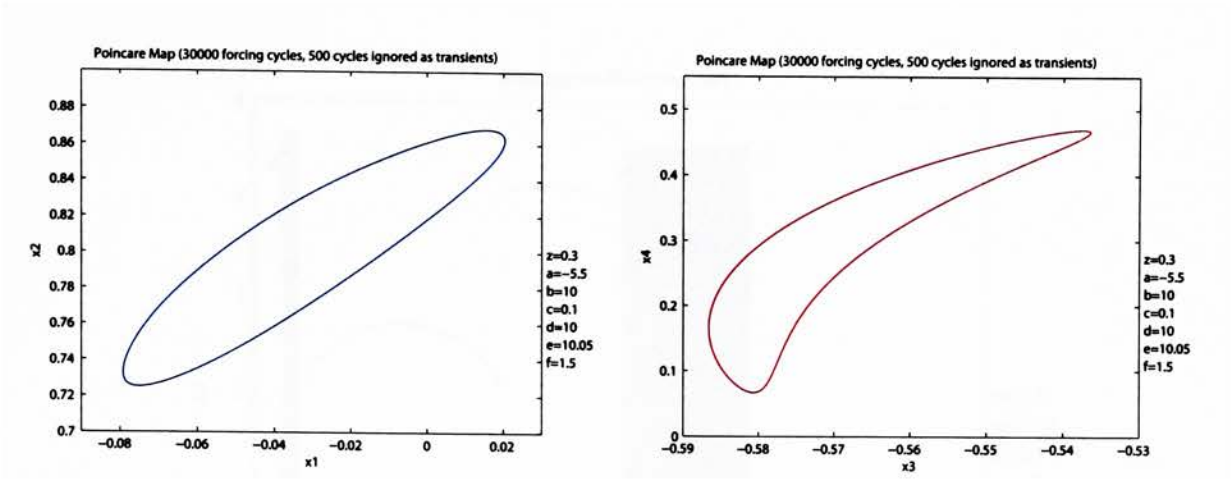


Figure 6.40: Lyapunov spectrum for positive e scan in a two well potential



(a) (b)

Figure 6.41: Quasiperiodic Poincaré maps for $e = 10.05$ in a two well potential corresponding to; (a) x_1–x_2 plane; (b) x_3–x_4 plane.

exponent. A period–2 orbit occurs after the chaos–exiting quasiperiodicity. The period–2 solution undergoes a period–doubling and halving sequence. The resumed period–2 solution is then interrupted by a small band of chaos. Crisis causes the solution to become period–1. The period–1 solution is interrupted by quasiperiodicity, which again is both a precursor to, and the resulting exit of, chaotic behavior. A second Lyapunov exponent also becomes positive and exhibits near zero behavior. The last two chaotic regions exit through crisis events. Both of these regions are also characterized by a second positive Lyapunov exponent. The scan ends with a well–entrained period–1 orbit.

6.3.2 Four Well Potential

The next scan of the e parameter occurred in a four well potential defined by $a = -1.5$, $b = 0.1$, $c = -2$, and $d = 0.5$. Similar to the synchronous forcing case, all of the interesting behavior occurred for values of e less than 1. The parameter e was rescanned with a finer step size of 0.001 from 0 to 0.9, the results of which are shown in Figures 6.42–6.44. The solution initially is well–entrained in one of the x-axis wells before a rapid period–doubling sequence results in four well chaos from $e = 0.029$ to $e = 0.054$. The maximal Lyapunov exponent associated with this chaos is 0.26. The Poincaré maps for one of the chaotic attractors in this region are shown in Figure 6.45. This chaotic region

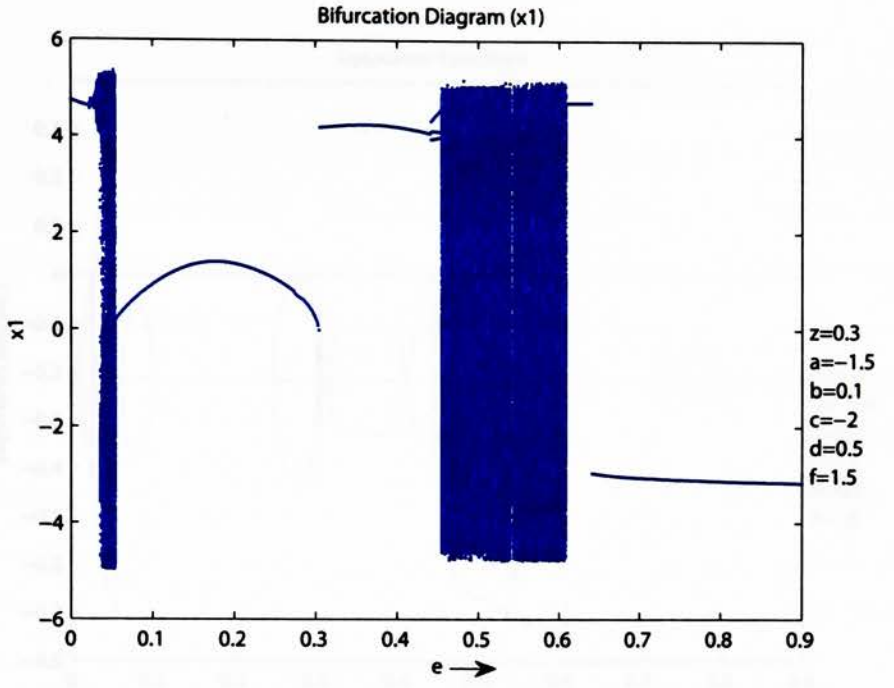


Figure 6.42: Bifurcation diagram for positive e scan in a four well potential (x_1)

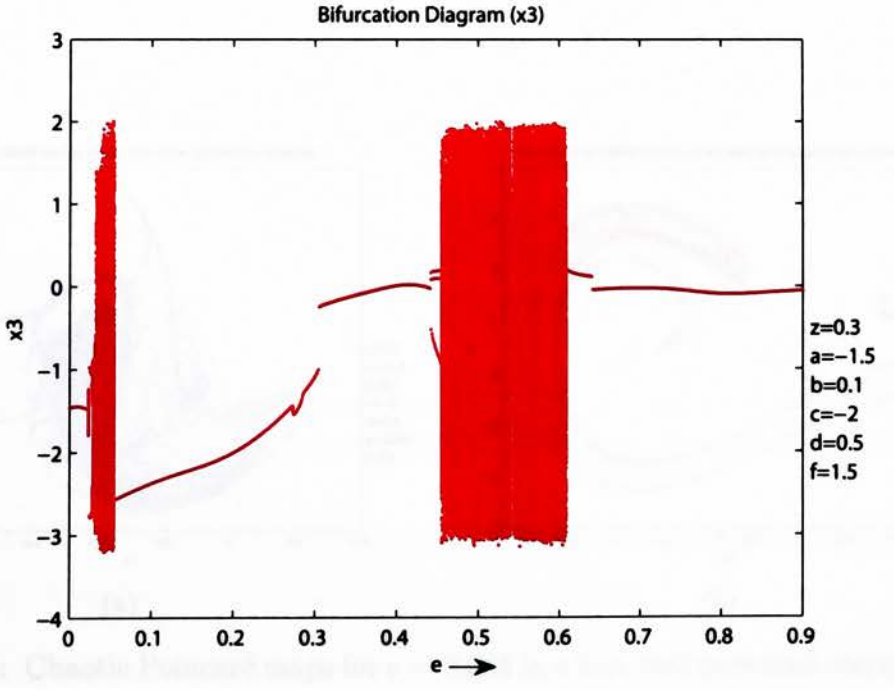


Figure 6.43: Bifurcation diagram for positive e scan in a four well potential (x_3)

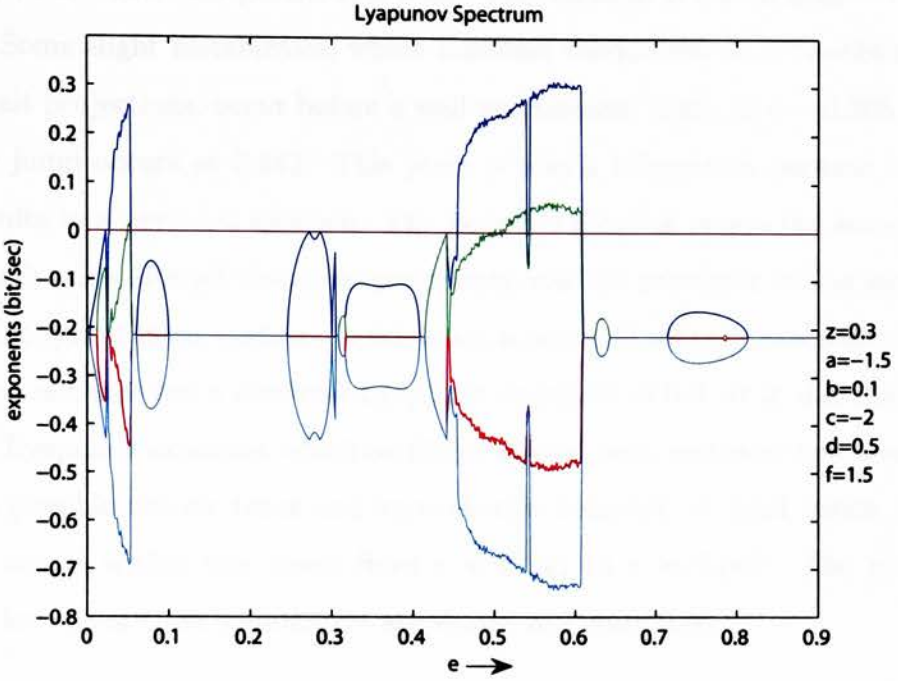
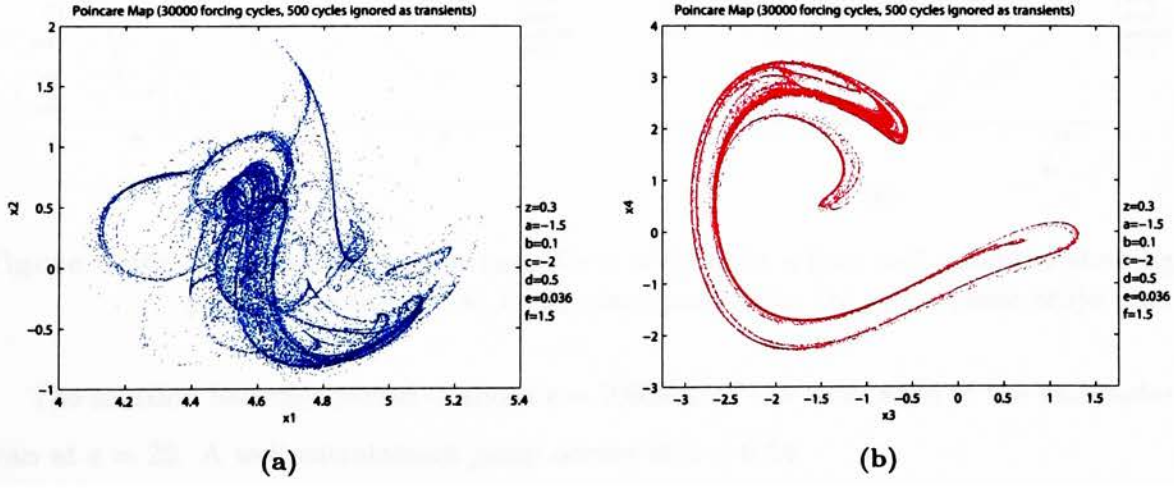


Figure 6.44: Lyapunov spectrum for positive e scan in a four well potential

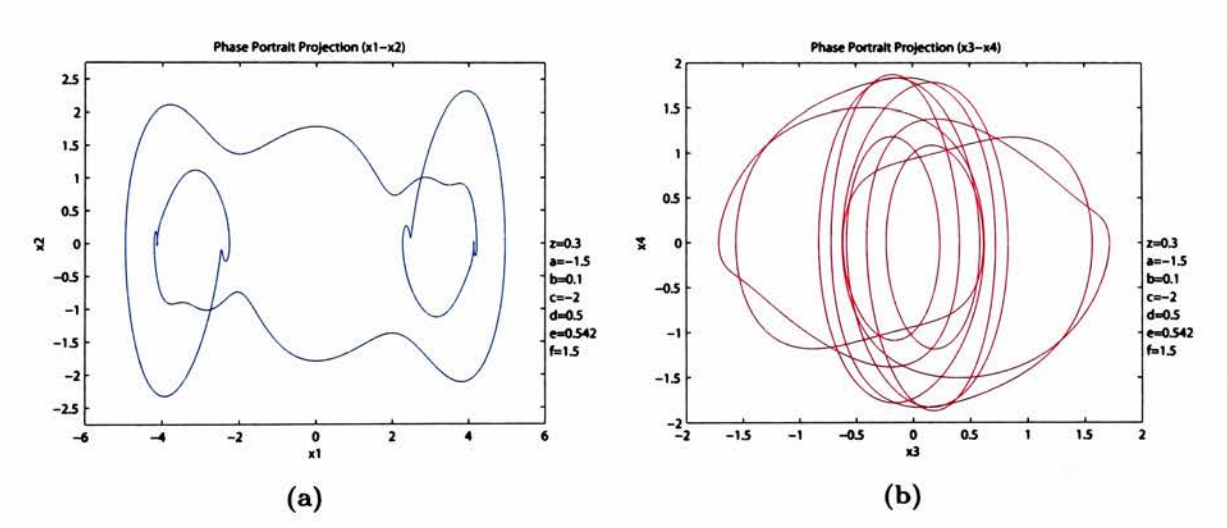


(a) **(b)**

Figure 6.45: Chaotic Poincaré maps for $e = 0.036$ in a four well potential corresponding to; (a) x_1–x_2 plane; (b) x_3–x_4 plane.

falls victim to a crisis event rendering a high power period–1 solution similar to that encountered in the four well potential a scan. The solution is stable from $e = 0.055$ to $e = 0.304$. Some slight instabilities, which manifest themselves as a "wobbling" of the phase portrait projections, occur before a well entrainment jump at $e = 0.305$.

Another jump occurs at 0.442. This jump is also a bifurcation because a period–1 solution results in a period–3 solution. The period–3 solution enters the second chaotic region through crisis. Recall that quasiperiodicity was the precursor to the second band of chaos in the synchronous system. In this scan, a period–3 solution precedes chaos. The second region of chaos has a maximal Lyapunov exponent of 0.3. It is also characterized by a second Lyapunov exponent which oscillates around zero, and later becomes positive, signifying a possible chaotic torus and hyperchaotic behavior. A brief region of period–5 behavior occurs within this chaos from $e = 0.541$ to $e = 0.543$. The phase plane projections for one of these parameters are shown in Figure 6.46.



(a) (b)

Figure 6.46: Phase portrait projections for $e = 0.542$ in a four well potential showing period–5 behavior; (a) x_1–x_2 plane projection; (b) x_3–x_4 plane projection.

The solution becomes period–1 above $e = 0.609$ until the conclusion of the parameter scan at $e = 20$. A well entrainment jump occurs at $e = 0.64$.

6.4 Small Nonlinearities

All of the parameter combinations listed in Section 5.5 also exhibited period–1 solutions over the entire parameter space considered. The out of phase forcing did not appear to qualitatively affect any of the systems with relatively small nonlinearities.

Chapter 7

Conclusions

7.1 Methods Summary

The methods, tools, and implementations discussed in Chapters 2 and 4 were successful in identifying the behaviors generated by the studied coupled Duffing oscillator system. Additionally each one of these methods is able to both independently verify the observed behavior as well as validate the other methods. The results obtained from four separate analysis methods, Lyapunov exponents, Poincaré maps, power spectra estimates, and phase portrait projections, were more than capable of either confirming or denying the existence of nonlinear phenomena.

The use of the animation capabilities avoided countless hours of tedious analysis. Consider that each one of the over 66,000 distinct simulations would generate two Poincaré maps, two power spectra, and two phase portrait projections. This would result in nearly 400,000 different plots to observe independently. For a meager amount of added computation time, 1000 separate simulations could be viewed in six animations with lengths of approximately one minute each. Add this to the time needed to examine two bifurcation diagrams and one Lyapunov spectrum, and the result does not even compare with the time that would be required to examine each figure individually. Some may argue that the bifurcation diagrams and Lyapunov spectra could be used exclusively for adequate analysis methods. However, the level of detail gained by observing the animations far outstrips these diagrams alone. The ability to pause, replay, and fast forward were used extensively to further examine the details of the developing solution paths encountered

in this study. At the time of printing, none of the literature available indicates that these animation analysis methods are currently, or have been previously employed.

The one drawback to the animations was the rather primitive scaling techniques. It was often difficult to observe regions of high intricacy if another larger amplitude result was located relatively closely in parameter space. Because the scaling acted to expand around the largest geometry in the plot, small intricate geometries were dwarfed by the relative scaling. This could easily be improved with a more complicated scaling algorithm that creates "zooming" effects through incrementally increased or decreased scaling in each frame.

7.2 Analysis Summary

Overall both of the studied systems exhibited a rich variety of nonlinear behavior through the range of investigated parameters. The majority of the scans of parameters a and b had entrances to, and exits from chaotic regions through period–doubling and crisis events. Quasiperiodicity was either very limited or nonexistent in these cases.

The parameter scans for e differed from the a and b scans with the emergence of extended regions of quasiperiodicity. It was found that quasiperiodicity replaced period–doubling and halving as transitionary behavior around chaotic regions. These regions of quasiperiodicity were almost always accompanied by period–3 behavior. The most extreme instance of which is the apparent reordering of quasiperiodicity to the period–3 solution in the synchronous system for the two well potential e scan. It was also found in the e scans that the possibility exists for hyperchaotic and chaotic torus solutions. A much more detailed study which attempted to exacerbate these behaviors would be required before their existences can be positively confirmed.

The maximal Lyapunov exponent calculated in any of the simulations was approximately 0.35. Most often a non–positive spike in the Lyapunov spectrum occurred near jumps in the solution path. In certain instances, analysis suggested that some of these jumps may have been caused by chaotic attractors located elsewhere in the phase space. The influence of these attractors may be sufficient to cause certain solution branches to

lose stability. Chaos was not found in any system that had positive values for both a and c. Thus chaos does not appear to occur unless at least two wells exist within the potential field.

A common result of well entrainment was a significant drop in the power associated with the signal. This drop occurred very abruptly. Solutions that began well–entrained typically did not exhibit abrupt positive gains in the signal power. The process of returning to the former full signal strength was gradual.

7.3 Potential Discrepancies

The lack of well defined hysteresis loops seems to be contrary to the amount of irreversibilities that exist within the system for many parameter scans. This may have several contributing factors including the complexity of the system and the parameter step size. Most of the "textbook" hysteresis examples occur in systems of low order with a small number of nonlinear terms. This investigated system was not only fourth–order, but contained a coupled cubic term. The parameter step size is also an unfortunate requirement of investigating solution branches. Varying the parameter continuously would be ideal, but this is impossible. A very small step is impractical due to the increased amount of simulations required, but a larger step may be such that it causes the system to enter another basin of attraction. Tufillaro discussed this in his study [31].

The variety of inter–verification analysis methods certainly reduces the risk of solutions being "misdiagnosed." Indeed, it was found that a potentially improper result in which chaotic behavior was mistakenly identified, was caught through the use of the multiple analysis methods. However, it may be the case that this occurs for more simulations within the entire parameter scan. The only way to reduce this potential effect is to increase the amount of simulation time, which in turn increases the computation time and data sizes. The sizes of the transient regions can be increased if a longer amount of data exists to disregard.

7.4 Future Work

Even though a large amount of parameter combinations were simulated, they represent a very small region of the five–dimensional parameter space defined by the potential field terms a, b, c, d, and e. This five–dimensional space is even a subspace of the more general seven–dimensional parameter space that also includes the damping and forcing amplitude z and f. Increasing dimensional variances can also exist where the damping or forcing amplitude are not equal in both oscillators. More simulations could be performed to include these additional parameters.

Several areas in this study identified certain ranges over the parameter space where it was suspected that multiple attracting solutions exist. The choice of initial conditions that resulted in these competing attracting solutions could be varied. Paul Raj, Rajasekar, and Murali conducted a limited basin of attraction study based on certain parameters [39]. A much more detailed basin of attraction study could be performed which identified the initial conditions that led to the differing attractors.

The bifurcation method used in this study is called the "brute–force" method by Parker and Chua [9]. One consequence of this method is that only stable solution branches are identified. This was a contributing factor to the discrepancy with irreversibilities and hysteresis loops. Another method called *continuation* would be able to identify all solution branches, including the unstable branches [9]. This method requires the numerical integration of another system of equations. It is unclear at this point whether a Lie series approach could be valid. If unstable solution branches were found to connect the stable branches identified throughout this study, it would indicate the existence of more hysteresis loops.

The methods outlined in this thesis could also be applied to verify an experimental analysis. A very similar system to the one depicted in Figure 3.2 could be created that uses a slender rod instead of a beam. Multiple wells, corresponding to a variety of potential fields, can be created using varying number of magnets located along two axes rather than one. The results of this experimental analysis could be compared directly with a theoretical model using experimental derived values for the equation parameters.

Detailed studies could be performed to more rigorously identify under which conditions hyperchaos and chaotic tori occur in this system. This would involve more parameter variations and basin of attraction studies. These phenomena were only found to occur in scans of the coupling parameter, which suggests that they are controlled through this parameter. However, it is possible that they exist elsewhere in the parameter space.

Most importantly this thesis discussed methods for a detailed parameter study that could generally be applied to any nonlinear system. The ever increasing speed of computer processors and increasing size of data storage predict that even more complex coupled systems will be able to be analyzed in the future.

Bibliography

[1] S. H. Strogatz. *Nonlinear Dynamics and Chaos.* Perseus Books, Cambridge, Massachusetts, 1994.

[2] F. C. Moon. *Chaotic and Fractal Dynamics.* John Wiley and Sons, New York, 1992.

[3] J. S. Török. Symbolic computation of Lie series for the solution of differential equations. In *ASEE Annual Conference Proceedings*, volume 2, pages 1934–1937, New Orleans, 1991.

[4] J. S. Török and S. H. Advani. Continuous transformation groups and series solution of initial value problems. *International Journal of Non-Linear Mechanics*, 20(4):283–289, 1985.

[5] Stanly Steinberg. Lie series and nonlinear ordinary differential equations. *Journal of Mathematical Analysis and Applications*, 101:39–63, 1984.

[6] A. Vasilik. Numerical solution of ordinary differential equations using Lie series. MS project with paper, Rochester Institute of Technology, 2002.

[7] A. Dick. Identification and prediction of nonlinear dynamics. Master's thesis, Rochester Institute of Technology, 2003.

[8] J. S. Török. A constructive methods for locating periodic solutions of linear and nonlinear systems. In *Dynamics and vibration of time-varying systems and structures*, volume 56, pages 13–17, Albuquerque, New Mexico, 1993. ASME Design Engineering Division.

[9] T. S. Parker and L. O. Chua. *Practical Numerical Algorithms for Chaotic Systems.* Springer-Verlag, New York, 1989.

[10] Y. Ueda. *The Road to Chaos.* Ariel Press, Santa Cruz, California, 1992.

[11] A. Wolf, J. B. Swift, H. L. Swinney, and J. A. Vastano. Determining Lyapunov exponents from a time series. *Physica D: Nonlinear Phenomena*, 16:285–317, 1985.

[12] J. S. Török. Estimation of Lyapunov exponents using a semi-discrete formulation. *Applied Mechanics Reviews*, 11(2):229–233, 1993.

[13] S. K. Mitra. *Digital Signal Processing*, chapter 7, 11, pages 454, 771–775. McGraw-Hill, New York, 2001.

[14] P. D. Welch. The use of fast Fourier transform for the estimation of power spectra: A method based on time averaging over short modified periodograms. *IEEE Transactions on Audio and Electroacoustics*, AU-17:70–73, 1967.

[15] V. Brunsden, J. Cortell, and P. J. Holmes. Power spectra of chaotic vibrations of a buckled beam. *Journal of Sound and Vibration*, 130(1):1–25, 1989.

[16] F. C. Moon and P. J. Holmes. A magnetoelastic strange attractor. *Journal of Sound and Vibration*, 65(2):275–296, 1979.

[17] J. Kozlowski, U. Parlitz, and W. Lauterborn. Bifurcation analysis of two coupled periodically driven Duffing oscillators. *Physical Review E*, 51(3):1861–1867, 1995.

[18] J. J. Stagliano, Jr, J. Wersinger, and E. E. Slaminka. Doubling bifurcations of destroyed $\mathbb{T}^2$ tori. *Physica D: Nonlinear Phenomena*, 92:164–177, 1996.

[19] H. Yin, J. Dai, and H. Zhang. Phase effect of two coupled periodically driven Duffing oscillators. *Physical Review E*, 58(5):5683–5688, 1998.

[20] D. E. Musielak, Z. E. Musielak, and J. W. Benner. Chaos and routes to chaos in coupled Duffing oscillators with multiple degrees of freedom. *Chaos, Solitons, and Fractals*, 24:907–922, 2005.

[21] J. W. Benner. *An investigation of the routes to chaos for complex nonlinear dynamical systems.* PhD thesis, University of Alabama at Huntsville, 1997.

[22] D. W. Martin. Decay rates of piano tones. *The Journal of the Acoustical Society of America*, 19(4):535–541, 1947.

[23] G. Weinreich. Coupled piano strings. *The Journal of the Acoustical Society of America*, 62(6):1474–1484, 1977.

[24] J. A. Elliot. Intrinsic nonlinear effect in vibrating strings. *American Journal of Physics*, 48(6):478–480, 1980.

[25] G. F. Carrier. On the non-linear vibration problem of the elastic string. *Quarterly of Applied Mathematics*, 3:157–165, 1945.

[26] J. W. Miles. Stability of forced oscillations of a vibrating string. *The Journal of the Acoustical Society of America*, 38:885–861, 1965.

[27] G. V. Anand. Large-amplitude damped free vibration of a stretched string. *The Journal of the Acoustical Society of America*, 45(5):1089–1096, 1969.

[28] R. Narisimha. Non-linear vibration of an elastic string. *Journal of Sound and Vibration*, 8(1):134–146, 1968.

[29] J. A. Elliot. Nonlinear resonance in vibrating strings. *American Journal of Physics*, 50(12):1148–1150, 1982.

[30] M. G. Olsson. The precessing spherical pendulum. *American Journal of Physics*, 46(11):1118–1119, 1978.

[31] N. B. Tufillaro. Nonlinear and chaotic string vibrations. *American Journal of Physics*, 57(5):408–414, 1989.

[32] S. A. Nayfeh and A. H. Nayfeh. Nonlinear interaction between two widely spaced modes—external excitation. *International Journal of Bifurcation and Chaos*, 3(2):417–427, 1993.

[33] R. Nabergoj, A. Tondl, and Z. Virag. Autoparametric resonance in an externally excited system. *Chaos, Solitons, and Fractals*, 4(2):263–273, 1994.

[34] G. M. Mahmoud, A. A. Mohamed, and S. A. Aly. Strange attractors and chaos control in periodically forced complex Duffing oscillators. *Physica A: Statistical Mechanics and its Applications*, 292:193–206, 2001.

[35] F. C. N. Lim, M. P. Cartmell, A. Cardoni, and M. Lucas. A preliminary investigation into optimising the response of vibrating systems used for ultrasonic cutting. *Journal of Sound and Vibration*, 272:1047–1069, 2004.

[36] S. Rajasekar and S. Paul Raj. The Painlevé property, integrability and chaotic behavior of a two-coupled Duffing oscillators. *Pramana-Journal of Physics*, 47(3):193–198, September 1996.

[37] S. Paul Raj and S. Rajasekar. Migration control in two coupled Duffing oscillators. *Physical Review E*, 55(5):6237–6240, 1997.

[38] S. Rajasekar, M. C. Valsakumar, and S. Paul Raj. Noise–induced jumps in two coupled Duffing oscillators. *Physica A: Statistical Mechanics and its Applications*, 261:417–434, 1998.

[39] S. Paul Raj, S. Rajasekar, and K. Murali. Coexisting chaotic attractors, their basin of attractions and synchronization of chaos in two coupled Duffing oscillators. *Physics Letters A*, 264:283–288, 1999.

Appendix A

Maple Programs

Maple program for calculating the powers of the infinitesimal generator U and its derivatives.

```
> restart;
> with(linalg):with(codegen,C):
> F1:=x2:
> F2:=f*cos(x5)-z*x2-a*x1-b*x1^3-e*x1*x3^2:
> F3:=x4:
> F4:=f*cos(x5)-z*x4-c*x3-d*x3^3-e*x1^2*x3:
> F5:=1:
> N:=6:
> for p from 1 to N do
>  vars:=[x1,x2,x3,x4,x5]:
>  vec:=array([F1,F2,F3,F4,F5]):
>  x:=array([x1,x2,x3,x4,x5]):
>  z1[p]:=x1:
>  z2[p]:=x2:
>  z3[p]:=x3:
>  z4[p]:=x4:
>  z5[p]:=x5:
>  for i from 1 to N do
>   J:=jacobian(x,vars):
>   Ux:=evalm(J&*vec):
>   z1[i]:=Ux[1]:
>   z2[i]:=Ux[2]:
>   z3[i]:=Ux[3]:
>   z4[i]:=Ux[4]:
>   z5[i]:=Ux[5]:
>   x:=Ux:
>  od:
> od:
> U:=array(1..N,1..5):
> dU:=array(1..N,1..5,1..5):
> for i from 1 to p do
```

```
32 >  U[i,1]:=z1[i]:
33 >  U[i,2]:=z2[i]:
34 >  U[i,3]:=z3[i]:
35 >  U[i,4]:=z4[i]:
36 >  U[i,5]:=z5[i]:
37 >  for k from 1 to 5 do
38 >    for m from 1 to 5 do
39 >    dU[i,k,m]:=diff(U[i,k],vars[m]);
40 >   od:
41 >  od:
42 > od:
43 > C(U,optimized,declarations='U::array(1..N,1..5)'):
44 > C(dU,optimized,declarations='dU::array(1..N,1..5,1..5)');
45 > all:=array(1..N+1,1..6,1..5):
46 > for i from 1 to N do
47 >  all[1,i,1]:=z1[i]:
48 >  all[1,i,2]:=z2[i]:
49 >  all[1,i,3]:=z3[i]:
50 >  all[1,i,4]:=z4[i]:
51 >  all[1,i,5]:=z5[i]:
52 > od:
53 > for i from 2 to N+1 do
54 >  for k from 1 to 5 do
55 >    for m from 1 to 5 do
56 >    all[i,k,m]:=diff(all[1,i-1,k],vars[m]);
57 >   od:
58 >  od:
59 > od:
60 > C(all,optimized,declarations='all::array(1..N+1,1..6,1..5)');
```

Appendix B

C Programs

In Appendices B and C, the symbol ↪ represents lines of code wrapped over multiple lines of text.

B.1 Lie Series and Lyapunov Exponents

```
/*-----------------------------------------------------------------------
 * Lyapunov exponent calculation and simulation function for nonlinearly
 *  coupled Duffing oscillators with synchronous forcing (syncmex.c)
 * By Joseph O'Day 2005
 *-----------------------------------------------------------------------
 * Uses BLAS functions, must compile with appropriate libraries
 *  mex <filename> MATLABPATH\extern\win32\lcc\libmwlapack.lib
 *-----------------------------------------------------------------------
 *  Input Parameters:
 *      Name:       Type:       Connotation:
 *      -----       -----       ------------
 *      TIME        mxArray*    stop time of simulation
 *      STEP        mxArray*    time step size
 *      INIT_CONS   mxArray*    array of initial conditions
 *      IGNORE      mxArray*    time to ignore before Lyapunov exponent
 *                                  calculation
 *      Z           mxArray*    equation variable (damping)
 *      A           mxArray*    equation variable (linear restoring)
 *      B           mxArray*    equation variable (cubic restoring)
 *      C           mxArray*    equation variable (linear restoring)
 *      D           mxArray*    equation variable (cubic restoring)
 *      E           mxArray*    equation variable (coupling strength)
 *      F           mxArray*    equation variable (forcing amplitude)
 *      SIMFILE     mxArray*    Matlab string for sim datafile name
 *      EXPFILE     mxArray*    Matlab string for exp datafile name
 *-----------------------------------------------------------------------*/
#include "mex.h"
#include "mat.h"
```

```
29 #include <math.h>
30 #include <float.h>
31 #include <stdlib.h>
32
33 #define TIME        prhs[0]
34 #define STEP        prhs[1]
35 #define INIT_CONS   prhs[2]
36 #define IGNORE      prhs[3]
37 #define Z           prhs[4]
38 #define A           prhs[5]
39 #define B           prhs[6]
40 #define C           prhs[7]
41 #define D           prhs[8]
42 #define E           prhs[9]
43 #define F           prhs[10]
44 #define SIMFILE     prhs[11]
45 #define EXPFILE     prhs[12]
46
47 #define APXORDER    6
48 #define SYSORDER    5
49 #define BASE        2
50 #define ZERO        1e-6
51
52 /* Function prototypes */
53 void syncmex(double step,double ICs[],int size,int iter,double z,
54              double a,double b,double c,double d,double e,double f,
55              double x1[],double x2[],double x3[],double x4[],double x5[],
56              double exp1[],double exp2[],double exp3[],double exp4[],
57              double exp5[],double *Er,double *Ea);
58 void lieapprox(double step,double z,double a,double b,double c,double d,
59                double e,double f,double x1,double x2,double x3,double x4,
60                double x5,double NX[],int lyapswitch,double DF[][SYSORDER]);
61 void gramschimidt(double DF[][SYSORDER],double BMAT[][SYSORDER],
62                   double mags[]);
63
64 /* BLAS function prototypes
65    See http://www.netlib.org/blas/ for details */
66 extern void dgemm(char*,char*,int*,int*,int*,double*,double*,int*,
67                   double*,int*,double*,double*,int*);
68 extern double dnrm2(int*,double*,int*);
69 extern double ddot(int*,double*,int*,double*,int*);
70 extern void dscal(int*,double*,double*,int*);
71 extern void daxpy(int*,double*,double*,int*,double*,int*);
72
73 /*-------------------------------------------------------------------
74  * Matlab Mex Gateway Function
75  *-------------------------------------------------------------------
```

```
76  *  Variables:
77  *      Name:         Type:       Connotation:
78  *      -----         -----       ------------
79  *      time          double      stop time of simulation
80  *      step          double      step size for Lie approx and Jacobian
81  *      ICs           double*     pointer to INIT_CONS
82  *      ignore        double      time to ignore before Lyapunov exponent
83  *                                    calculation
84  *      z             double      equation variable (damping)
85  *      a             double      equation variable (linear restoring)
86  *      b             double      equation variable (cubic restoring)
87  *      c             double      equation variable (linear restoring)
88  *      d             double      equation variable (cubic restoring)
89  *      e             double      equation variable (coupling strength)
90  *      f             double      equation variable (forcing amplitude)
91  *      size          int         array size of output variables
92  *      iter          int         array size of Lyapunov exponents
93  *      mxX1          mxArray*    Matlab array of state x1
94  *      mxX2          mxArray*    Matlab array of state x2
95  *      mxX3          mxArray*    Matlab array of state x3
96  *      mxX4          mxArray*    Matlab array of state x4
97  *      mxX5          mxArray*    Matlab array of state x5
98  *      x1            double*     pointer to mxX1
99  *      x2            double*     pointer to mxX1
100 *      x3            double*     pointer to mxX1
101 *      x4            double*     pointer to mxX1
102 *      x5            double*     pointer to mxX1
103 *      mxEXP1        mxArray*    Matlab array of Lyapunov exponent 1
104 *      mxEXP2        mxArray*    Matlab array of Lyapunov exponent 2
105 *      mxEXP3        mxArray*    Matlab array of Lyapunov exponent 3
106 *      mxEXP4        mxArray*    Matlab array of Lyapunov exponent 4
107 *      mxEXP5        mxArray*    Matlab array of Lyapunov exponent 5
108 *      mxER          mxArray*    relative convergence
109 *      mxEA          mxArray*    absolute convergence
110 *      exp1          double*     pointer to mxEXP1
111 *      exp2          double*     pointer to mxEXP2
112 *      exp3          double*     pointer to mxEXP3
113 *      exp4          double*     pointer to mxEXP4
114 *      exp5          double*     pointer to mxEXP5
115 *      Er            double*     pointer to mxER
116 *      Ea            double*     pointer to mxEA
117 *      mxFINALEXP1 mxArray*      final value of lyapunov exponent 1
118 *      mxFINALEXP2 mxArray*      final value of lyapunov exponent 2
119 *      mxFINALEXP3 mxArray*      final value of lyapunov exponent 3
120 *      mxFINALEXP4 mxArray*      final value of lyapunov exponent 4
121 *      mxFINALEXP5 mxArray*      final value of lyapunov exponent 5
122 *      buflen        int         length of string buffer for file name
```

```
123  *       simfile     char*       C filename string
124  *       expfile     char*       C filename string
125  *       status      int         function call status variable
126  *       simmat      MATfile*    mat-file pointer
127  *       expmat      MATfile*    mat-file pointer
128  *       mxSIMNAME   mxArray*    filename for storage in data file
129  *       mxEXPNAME   mxArray*    filename for storage in data file
130  *-------------------------------------------------------------------*/
131 void mexFunction( int nlhs, mxArray *plhs[],
132                   int nrhs, const mxArray *prhs[] )
133 {
134     double  time,step,*ICs,z,a,b,c,d,e,f,ignore,*x1,*x2,*x3,*x4,*x5,
135             *exp1,*exp2,*exp3,*exp4,*exp5,*Er,*Ea;
136     int     ncols,size,iter,buflen,status;
137     char    *simfile,*expfile;
138     mxArray *mxX1,*mxX2,*mxX3,*mxX4,*mxX5,*mxEXP1,*mxEXP2,*mxEXP3,
139             *mxEXP4,*mxEXP5,*mxER,*mxEA,*mxSIMNAME,*mxEXPNAME,
140             *mxFINALEXP1,*mxFINALEXP2,*mxFINALEXP3,*mxFINALEXP4,
141             *mxFINALEXP5;
142     MATFile *simmat,*expmat;
143
144     /* check for input. */
145     if(nrhs != 13)
146         mexErrMsgTxt("\n 13 inputs required.\n\t - time\n\t - step size
147 ↪ \n\t - initial conditions (5)\n\t - time to ignore before Lyapunov
148 ↪ calculation\n\t - 7 equation parameters\n\t - 2 filenames");
149
150     /* Get inputs */
151     time = mxGetScalar(TIME);
152     step = mxGetScalar(STEP);
153     ICs = mxGetPr(INIT_CONS);
154     ignore = mxGetScalar(IGNORE);
155     z = mxGetScalar(Z);
156     a = mxGetScalar(A);
157     b = mxGetScalar(B);
158     c = mxGetScalar(C);
159     d = mxGetScalar(D);
160     e = mxGetScalar(E);
161     f = mxGetScalar(F);
162
163     /* Get the dimensions of the matrix input initial conditions */
164     ncols = mxGetN(INIT_CONS);
165     if(ncols != 5)
166         mexErrMsgTxt("\n 5 ICs required!");
167
168     /* Calculate data array sizes */
169     size = ceil(time/step);
```

```
170     iter = ceil((time-ignore)/step);
171     if(time<ignore)
172         mexErrMsgTxt("\n Simulation time must be larger than ignored
173 ↪ transient data!");
174
175     /* Create and reference data arrays */
176     mxX1 = mxCreateDoubleMatrix(size+1,1,mxREAL);
177     mxX2 = mxCreateDoubleMatrix(size+1,1,mxREAL);
178     mxX3 = mxCreateDoubleMatrix(size+1,1,mxREAL);
179     mxX4 = mxCreateDoubleMatrix(size+1,1,mxREAL);
180     mxX5 = mxCreateDoubleMatrix(size+1,1,mxREAL);
181     mxEXP1 = mxCreateDoubleMatrix(iter+1,1,mxREAL);
182     mxEXP2 = mxCreateDoubleMatrix(iter+1,1,mxREAL);
183     mxEXP3 = mxCreateDoubleMatrix(iter+1,1,mxREAL);
184     mxEXP4 = mxCreateDoubleMatrix(iter+1,1,mxREAL);
185     mxEXP5 = mxCreateDoubleMatrix(iter+1,1,mxREAL);
186     mxER = mxCreateDoubleMatrix(1,1,mxREAL);
187     mxEA = mxCreateDoubleMatrix(1,1,mxREAL);
188     x1 = mxGetPr(mxX1);
189     x2 = mxGetPr(mxX2);
190     x3 = mxGetPr(mxX3);
191     x4 = mxGetPr(mxX4);
192     x5 = mxGetPr(mxX5);
193     exp1 = mxGetPr(mxEXP1);
194     exp2 = mxGetPr(mxEXP2);
195     exp3 = mxGetPr(mxEXP3);
196     exp4 = mxGetPr(mxEXP4);
197     exp5 = mxGetPr(mxEXP5);
198     Er = mxGetPr(mxER);
199     Ea = mxGetPr(mxEA);
200
201     /* Call the C subroutine SYNCMEX.C */
202     syncmex(step,ICs,size,iter,z,a,b,c,d,e,f,x1,x2,x3,x4,x5,exp1,exp2,
203 ↪ exp3,exp4,exp5,Er,Ea);
204
205     /* Get final exponent values (faster loading for LYAPSPECMEX.C) */
206     mxFINALEXP1 = mxCreateDoubleScalar(exp1[iter]);
207     mxFINALEXP2 = mxCreateDoubleScalar(exp2[iter]);
208     mxFINALEXP3 = mxCreateDoubleScalar(exp3[iter]);
209     mxFINALEXP4 = mxCreateDoubleScalar(exp4[iter]);
210     mxFINALEXP5 = mxCreateDoubleScalar(exp5[iter]);
211
212     /* Get filenames */
213     buflen = (mxGetM(SIMFILE) * mxGetN(SIMFILE) * sizeof(mxChar)) + 1;
214     simfile = mxCalloc(buflen, sizeof(char));
215     status = mxGetString(SIMFILE, simfile, buflen);
216     if(status != 0)
```

```
217         mexErrMsgTxt("SIM filename error.");
218
219     buflen = (mxGetM(EXPFILE) * mxGetN(EXPFILE) * sizeof(mxChar)) + 1;
220     expfile = mxCalloc(buflen, sizeof(char));
221     status = mxGetString(EXPFILE, expfile, buflen);
222     if(status != 0)
223         mexErrMsgTxt("EXP filename error.");
224
225     mxSIMNAME = mxCreateString(simfile);
226     mxEXPNAME = mxCreateString(expfile);
227
228     /* Open files */
229     simmat = matOpen(simfile, "w");
230     if(simmat == NULL) {
231         printf("\n Error opening file: %s", simfile);
232         return; }
233     expmat = matOpen(expfile, "w");
234     if(simmat == NULL) {
235         printf("\n Error opening file: %s", expfile);
236         return; }
237
238     /* Write sim data file */
239     printf("\n Writing sim file: %s",simfile);
240     status = matPutVariable(simmat, "x1", mxX1);
241     if(status != 0) {
242         printf("\n Error writing x1 to %s",simfile);
243         return; }
244     else { mxDestroyArray(mxX1); }
245     status = matPutVariable(simmat, "x2", mxX2);
246     if(status != 0) {
247         printf("\n Error writing x2 to %s",simfile);
248         return; }
249     else { mxDestroyArray(mxX2); }
250     status = matPutVariable(simmat, "x3", mxX3);
251     if(status != 0) {
252         printf("\n Error writing x3 to %s",simfile);
253         return; }
254     else { mxDestroyArray(mxX3); }
255     status = matPutVariable(simmat, "x4", mxX4);
256     if(status != 0) {
257         printf("\n Error writing x4 to %s",simfile);
258         return; }
259     else { mxDestroyArray(mxX4); }
260     status = matPutVariable(simmat, "t", mxX5);
261     if(status != 0) {
262         printf("\n Error writing x5 to %s",simfile);
263         return; }
```

```
264       else { mxDestroyArray(mxX5); }
265       status = matPutVariable(simmat, "datafile", mxSIMNAME);
266       if(status != 0) {
267           printf("\n Error writing datafile to %s",simfile);
268           return; }
269       else { mxDestroyArray(mxSIMNAME); }
270
271       /* Close sim data file */
272       if(matClose(simmat) != 0) {
273           printf("Error closing file %s\n",simfile);
274           return; }
275
276       /* Write exp data file */
277       printf("\n Writing exp file: %s",expfile);
278       status = matPutVariable(expmat, "exp1", mxEXP1);
279       if(status != 0) {
280           printf("\n Error writing exp1 to %s",expfile);
281           return; }
282       else { mxDestroyArray(mxEXP1); }
283       status = matPutVariable(expmat, "exp2", mxEXP2);
284       if(status != 0) {
285           printf("\n Error writing exp2 to %s",expfile);
286           return; }
287       else { mxDestroyArray(mxEXP2); }
288       status = matPutVariable(expmat, "exp3", mxEXP3);
289       if(status != 0) {
290           printf("\n Error writing exp3 to %s",expfile);
291           return; }
292       else { mxDestroyArray(mxEXP3); }
293       status = matPutVariable(expmat, "exp4", mxEXP4);
294       if(status != 0) {
295           printf("\n Error writing exp4 to %s",expfile);
296           return; }
297       else { mxDestroyArray(mxEXP4); }
298       status = matPutVariable(expmat, "exp5", mxEXP5);
299       if(status != 0) {
300           printf("\n Error writing exp5 to %s",expfile);
301           return; }
302       else { mxDestroyArray(mxEXP5); }
303       status = matPutVariable(expmat, "Er", mxER);
304       if(status != 0) {
305           printf("\n Error writing Er to %s",expfile);
306           return; }
307       else { mxDestroyArray(mxER); }
308       status = matPutVariable(expmat, "Ea", mxEA);
309       if(status != 0) {
310           printf("\n Error writing Ea to %s",expfile);
```

```
311         return; }
312     else { mxDestroyArray(mxEA); }
313     status = matPutVariable(expmat, "finalexp1", mxFINALEXP1);
314     if(status != 0) {
315         printf("\n Error writing finalexp1 to %s",expfile);
316         return; }
317     else { mxDestroyArray(mxFINALEXP1); }
318     status = matPutVariable(expmat, "finalexp2", mxFINALEXP2);
319     if(status != 0) {
320         printf("\n Error writing finalexp2 to %s",expfile);
321         return; }
322     else { mxDestroyArray(mxFINALEXP2); }
323     status = matPutVariable(expmat, "finalexp3", mxFINALEXP3);
324     if(status != 0) {
325         printf("\n Error writing finalexp3 to %s",expfile);
326         return; }
327     else { mxDestroyArray(mxFINALEXP3); }
328     status = matPutVariable(expmat, "finalexp4", mxFINALEXP4);
329     if(status != 0) {
330         printf("\n Error writing finalexp4 to %s",expfile);
331         return; }
332     else { mxDestroyArray(mxFINALEXP4); }
333     status = matPutVariable(expmat, "finalexp5", mxFINALEXP5);
334     if(status != 0) {
335         printf("\n Error writing finalexp5 to %s",expfile);
336         return; }
337     else { mxDestroyArray(mxFINALEXP5); }
338     status = matPutVariable(expmat, "datafile", mxEXPNAME);
339     if(status != 0) {
340         printf("\n Error writing datafile to %s",expfile);
341         return; }
342     else { mxDestroyArray(mxEXPNAME); }
343
344     /* Close exp data file */
345     if(matClose(expmat) != 0) {
346         printf("Error closing file %s\n",expfile);
347         return;  }
348
349     printf("\n\n SYNCMEX.C TERMINATING ...\n");
350     return;
351 }
352
353 /*---------------------------------------------------------------------
354  * Main function (syncmex)
355  *---------------------------------------------------------------------
356  *  Variables:
357  *      Name:       Type:       Connotation:
```

```
 *       -----          -----        ------------
 *       DF             double[][]   Jacobian of system
 *       NX             double[]     array of next values of state variables
 *       BMAT           double[][]   basis vectors for state space (row-wise)
 *       mags           double[]     magnitudes of deformed basis vectors
 *       sum            double[]     running sum of vector magnitudes
 *       temp           double       temporary variable
 *       finexpptr      double*[]    array of pointers to last exponents
 *       KT             double       Lyapunov iterations multiplied by step size
 *       ei             int          Lyapunov counter, implemented to control
 *                                       size of transient data to be ignored
 *       i              int          counter
 *       j              int          counter
 *       k              int          counter
 *       lyapswitch     int          switch for Lyapunov exponent computation
 *-------------------------------------------------------------------------*/
void syncmex(double step, double ICs[SYSORDER], int size, int iter,
             double z, double a, double b, double c, double d, double e,
             double f, double x1[], double x2[], double x3[], double x4[],
             double x5[], double exp1[], double exp2[], double exp3[],
             double exp4[], double exp5[], double *Er, double *Ea)
{
    double DF[SYSORDER][SYSORDER]={0}, NX[SYSORDER]={0},
           BMAT[SYSORDER][SYSORDER]={{1},{0,1},{0,0,1},{0,0,0,1},{0,0,0,0,1}},
           mags[SYSORDER]={0},sum[SYSORDER]={0},KT=1,temp,
           *finexpptr[SYSORDER]=NULL;
    int    ei=0,i,j,k,lyapswitch=0;

    x1[0] = ICs[0];
    x2[0] = ICs[1];
    x3[0] = ICs[2];
    x4[0] = ICs[3];
    x5[0] = ICs[4];

    ei = size - iter;

    for (i=0;i<(size+1);i++) {
        /* Lyapunov calculation (one additional than time stepping) */
        if(i>=ei) {
            lyapswitch = 1;
            /* set/reset initial DF */
            DF[0][0]=1.0;DF[0][1]=0.0;DF[0][2]=0.0;DF[0][3]=0.0;DF[0][4]=0.0;
            DF[1][0]=0.0;DF[1][1]=1.0;DF[1][2]=0.0;DF[1][3]=0.0;DF[1][4]=0.0;
            DF[2][0]=0.0;DF[2][1]=0.0;DF[2][2]=1.0;DF[2][3]=0.0;DF[2][4]=0.0;
            DF[3][0]=0.0;DF[3][1]=0.0;DF[3][2]=0.0;DF[3][3]=1.0;DF[3][4]=0.0;
            DF[4][0]=0.0;DF[4][1]=0.0;DF[4][2]=0.0;DF[4][3]=0.0;DF[4][4]=1.0;
        }
```

```
405         lieapprox(step,z,a,b,c,d,e,f,x1[i],x2[i],x3[i],x4[i],x5[i],NX,
406 ↪ lyapswitch,DF);
407         if(i<size) {
408             x1[i+1] = NX[0];
409             x2[i+1] = NX[1];
410             x3[i+1] = NX[2];
411             x4[i+1] = NX[3];
412             x5[i+1] = NX[4];
413         }
414
415         if(i>=ei) {
416             j = i-ei;
417             gramschimidt(DF,BMAT,mags);
418             for (k=0;k<SYSORDER;k++) {
419                 sum[k] += log(mags[k])/log(BASE); }
420             KT = step*(j+1);
421             exp1[j] = sum[0]/KT;
422             exp2[j] = sum[1]/KT;
423             exp3[j] = sum[2]/KT;
424             exp4[j] = sum[3]/KT;
425             exp5[j] = sum[4]/KT;
426         }
427     }
428
429     /* Convergence */
430     finexpptr[0] = &exp1[iter];
431     finexpptr[1] = &exp2[iter];
432     finexpptr[2] = &exp3[iter];
433     finexpptr[3] = &exp4[iter];
434     finexpptr[4] = &exp5[iter];
435
436     for (i=0;i<SYSORDER;i++) {
437         if(fabs(*finexpptr[i]) > ZERO) {
438             temp = fabs(*finexpptr[i]-*(finexpptr[i]-1))/fabs(*(finexpptr[i]-1));
439             if(temp>*Er) { *Er = temp; }
440             temp = fabs(*finexpptr[i]-*(finexpptr[i] - 1));
441             if(temp>*Ea) { *Ea = temp; }
442         }
443     }
444 }
445
446 /*-----------------------------------------------------------------------
447  * Lie Series solution approximating function (lieapprox)
448  *-----------------------------------------------------------------------
449  *  Purpose:
450  *      Calculates the next value of system state variables and the
451  *          Jacobian via a 6th-order Lie Series approximation
```

```
452  *
453  *  Variables:
454  *       Name:        Type:            Connotation:
455  *       -----        -----            ------------
456  *       h            long double[]    array of step times to the power of
457  *                                         the approximating order
458  *       U            double[][]       characteristic infinitesimal operators
459  *       dU           double[][][]     derivatives of characteristic
460  *                                         infinitesimal operators
461  *       t#           double           Maple optimizing variables
462  *       i            int              counter
463  *       j            int              counter
464  *       k            int              counter
465  *-------------------------------------------------------------------*/
466  void lieapprox(double step, double z, double a, double b, double c,
467                 double d, double e, double f, double x1, double x2,
468                 double x3, double x4, double x5, double NX[SYSORDER],
469                 int lyapswitch, double DF[SYSORDER][SYSORDER])
470  {
471     long double h[APXORDER]={0};
472     double      U[APXORDER][SYSORDER]={0},
473                 dU[APXORDER][SYSORDER][SYSORDER]={0},t1,t2,t5,t8,t9,t11,
474                 t16,t18,t19,t20,t21,t22,t28,t29,t30,t34,t35,t36,t39,t40,
475                 t41,t43,t44,t45,t47,t49,t50,t53,t54,t57,t62,t63,t64,t69,
476                 t71,t72,t74,t75,t77,t79,t80,t81,t82,t83,t84,t85,t87,t88,
477                 t89,t90,t91,t92,t94,t95,t97,t98,t103,t105,t106,t107,t108,
478                 t110,t111,t112,t113,t114,t115,t116,t119,t120,t123,t124,
479                 t125,t128,t129,t130,t133,t134,t136,t137,t140,t141,t143,
480                 t144,t145,t146,t147,t148,t150,t151,t156,t158,t159,t160,
481                 t161,t163,t165,t167,t168,t169,t170,t171,t174,t175,t176,
482                 t177,t178,t179,t180,t181,t182,t183,t184,t185,t186,t187,
483                 t188,t189,t190,t191,t192,t193,t194,t196,t197,t198,t199,
484                 t200,t201,t208,t210,t211,t212,t214,t215,t216,t217,t218,
485                 t219,t220,t221,t222,t224,t225,t227,t228,t229,t230,t231,
486                 t232,t234,t235,t236,t237,t238,t239,t241,t242,t243,t244,
487                 t245,t246,t247,t248,t249,t252,t253,t255,t256,t258,t259,
488                 t260,t261,t262,t263,t264,t265,t266,t267,t269,t270,t272,
489                 t274,t275,t277,t279,t281,t283,t284,t285,t287,t288,t289,
490                 t290,t291,t292,t299,t301,t302,t303,t304,t305,t306,t307,
491                 t308,t310,t311,t313,t316,t317,t320,t321,t323,t324,t326,
492                 t327,t328,t329,t330,t331,t332,t333,t335,t336,t337,t340,
493                 t345,t346,t347,t348,t349,t350,t351,t353,t355,t356,t357,
494                 t358,t359,t360,t361,t362,t364,t365,t366,t367,t368,t369,
495                 t370,t371,t372,t373,t374,t376,t377,t378,t379,t380,t381,
496                 t383,t384,t385,t386,t387,t388,t389,t391,t392,t393,t394,
497                 t395,t396,t397,t398,t399,t400,t401,t402,t403,t404,t405,
498                 t406,t410,t413,t418,t422,t424,t426,t427,t428,t430,t431,
```

```
499                 t432,t433,t435,t436,t438,t439,t440,t441,t442,t444,t446,
500                 t447,t448,t449,t450,t451,t452,t453,t454,t455,t456,t457,
501                 t458,t459,t460,t461,t464,t466,t469,t470,t472,t475,t478,
502                 t479,t480,t481,t482,t483,t484,t485,t486,t488,t489,t490,
503                 t491,t492,t493,t494,t495,t497,t499,t500,t501,t502,t503,
504                 t505,t507,t508,t509,t510,t511,t512,t513,t514,t515,t517,
505                 t519,t520,t521,t524,t525,t526,t528,t529,t534,t536,t538,
506                 t539,t540,t542,t543,t545,t547,t548,t550,t552,t553,t556,
507                 t557,t559,t560,t561,t564,t569,t570,t571,t572,t573,t574,
508                 t575,t576,t577,t578,t579,t587,t590,t592,t593,t594,t595,
509                 t597,t600,t601,t602,t603,t604,t606,t607,t608,t609,t610,
510                 t611,t612,t613,t614,t615,t616,t617,t630,t631,t633,t642,
511                 t649,t651,t660,t661,t662,t664,t665,t666,t668,t670,t671,
512                 t674,t675,t683,t685,t694,t695,t699,t700,t711,t712,t714,
513                 t721,t722,t723,t725,t726,t730,t733,t735,t737,t738,t747,
514                 t749,t750,t752,t753,t755,t756,t759,t761,t767,t768,t770,
515                 t771,t773,t774,t775,t776,t778,t781,t783,t784,t786,t787,
516                 t788,t790,t791,t792,t794,t796,t798,t799,t800,t803,t806,
517                 t808,t809,t812,t814,t832,t834,t835,t838,t840,t853,t854,
518                 t858,t859,t861,t867,t877,t878,t879,t881,t884,t886,t889,
519                 t891,t892,t894,t896,t897,t898,t899,t900,t903,t906,t907,
520                 t912,t914,t918,t919,t920,t921,t924,t926,t931,t933,t934,
521                 t935,t949,t951,t955,t959,t980,t983,t995,t997,t1015,t1018,
522                 t1026,t1035,t1040,t1042,t1045,t1047,t1048,t1059,t1065,
523                 t1067,t1098,t1106,t1108,t1117,t1121,t1123,t1125,t1126,
524                 t1128,t1130,t1131,t1132,t1149,t1151,t1153,t1154,t1159,
525                 t1172,t1183;
526     int         i,j,k;
527
528     h[0] = step;
529     h[1] = pow(step,2.0)/2.0;
530     h[2] = pow(step,3.0)/6.0;
531     h[3] = pow(step,4.0)/24.0;
532     h[4] = pow(step,5.0)/120.0;
533     h[5] = pow(step,6.0)/720.0;
534
535     /* Maple optimizing variables for U */
536     t1 = cos(x5);
537     t2 = f*t1;
538     t5 = x1*x1;
539     t8 = e*x1;
540     t9 = x3*x3;
541     t11 = t2-z*x2-a*x1-b*t5*x1-t8*t9;
542     t16 = e*t5;
543     t18 = t2-z*x4-c*x3-d*t9*x3-t16*x3;
544     t19 = b*t5;
545     t20 = 3.0*t19;
```

```
546       t21 = e*t9;
547       t22 = -a-t20-t21;
548       t28 = sin(x5);
549       t29 = f*t28;
550       t30 = t22*x2-z*t11-2.0*t8*x3*x4-t29;
551       t34 = d*t9;
552       t35 = 3.0*t34;
553       t36 = -c-t35-t16;
554       t39 = -2.0*t8*x3*x2+t36*x4-z*t18-t29;
555       t40 = b*x1;
556       t41 = t40*x2;
557       t43 = z*t22;
558       t44 = e*x3;
559       t45 = t44*x4;
560       t47 = -6.0*t41-t43-2.0*t45;
561       t49 = z*z;
562       t50 = -a-t20-t21+t49;
563       t53 = z*e;
564       t54 = x1*x3;
565       t57 = -t44*x2+t53*t54-t8*x4;
566       t62 = z*f;
567       t63 = t62*t28;
568       t64 = t47*x2+t50*t11+2.0*t57*x4-2.0*t8*x3*t18+t63-t2;
569       t69 = t8*x2;
570       t71 = d*x3;
571       t72 = t71*x4;
572       t74 = z*t36;
573       t75 = -2.0*t69-6.0*t72-t74;
574       t77 = -c-t35-t16+t49;
575       t79 = 2.0*t57*x2-2.0*t8*x3*t11+t75*x4+t77*t18+t63-t2;
576       t80 = b*x2;
577       t81 = z*b;
578       t82 = t81*x1;
579       t83 = -t80+t82;
580       t84 = 6.0*t83*x2;
581       t85 = t40*t11;
582       t87 = t50*t22;
583       t88 = t53*x3;
584       t89 = e*x4;
585       t90 = t88-t89;
586       t91 = 2.0*t90*x4;
587       t92 = t44*t18;
588       t94 = e*e;
589       t95 = t94*t5;
590       t97 = 4.0*t95*t9;
591       t98 = t84-6.0*t85+t87+t91-2.0*t92+t97;
592       t103 = -12.0*t41-t43-4.0*t45-t50*z;
```

```
593       t105 = 2.0*t90*x2;
594       t106 = t44*t11;
595       t107 = 2.0*t106;
596       t108 = t50*e;
597       t110 = 2.0*t108*t54;
598       t111 = e*x2;
599       t112 = t53*x1;
600       t113 = -t111+t112;
601       t114 = 2.0*t113*x4;
602       t115 = t8*t18;
603       t116 = 2.0*t115;
604       t119 = 2.0*t8*x3*t36;
605       t120 = t105-t107-t110+t114-t116-t119;
606       t123 = t50*f;
607       t124 = t123*t28;
608       t125 = x3*f;
609       t128 = 2.0*t8*t125*t28;
610       t129 = t62*t1;
611       t130 = t98*x2+t103*t11+t120*x4+4.0*t57*t18-t124+t128+t129+t29;
612       t133 = 2.0*t8*x3*t22;
613       t134 = t77*e;
614       t136 = 2.0*t134*t54;
615       t137 = t105-t107-t133+t114-t116-t136;
616       t140 = 2.0*t113*x2;
617       t141 = t8*t11;
618       t143 = d*x4;
619       t144 = z*d;
620       t145 = t144*x3;
621       t146 = -t143+t145;
622       t147 = 6.0*t146*x4;
623       t148 = t71*t18;
624       t150 = t77*t36;
625       t151 = t140-2.0*t141+t97+t147-6.0*t148+t150;
626       t156 = -4.0*t69-12.0*t72-t74-t77*z;
627       t158 = t77*f;
628       t159 = t158*t28;
629       t160 = t137*x2+4.0*t57*t11+t151*x4+t156*t18+t128-t159+t129+t29;
630       t161 = t81*x2;
631       t163 = b*t11;
632       t165 = t40*t22;
633       t167 = t50*b;
634       t168 = t167*x1;
635       t169 = 6.0*t168;
636       t170 = t94*t9;
637       t171 = t170*x1;
638       t174 = (6.0*t161-6.0*t163-12.0*t165-t169+12.0*t171)*x2;
639       t175 = 12.0*t83*t11;
```

```
640       t176 = t103*t22;
641       t177 = t44*t22;
642       t178 = 2.0*t177;
643       t179 = t19*t44;
644       t180 = 12.0*t179;
645       t181 = t108*x3;
646       t182 = 2.0*t181;
647       t183 = t53*x4;
648       t184 = 2.0*t183;
649       t185 = e*t18;
650       t186 = 2.0*t185;
651       t187 = t95*x3;
652       t188 = 8.0*t187;
653       t189 = t44*t36;
654       t190 = 2.0*t189;
655       t191 = -t178+t180-t182+t184-t186+t188-t190;
656       t192 = t191*x4;
657       t193 = 4.0*t90*t18;
658       t194 = 4.0*t57*e;
659       t196 = 2.0*t194*t54;
660       t197 = t40*t29;
661       t198 = 6.0*t197;
662       t199 = t44*t29;
663       t200 = 2.0*t199;
664       t201 = t174+t175+t176+t192+t193-t196+t198+t200;
665       t208 = 12.0*t83*x2+t84-18.0*t85+t87+t91-6.0*t92+t97-t103*z+4.0*t90*x4;
666       t210 = t191*x2;
667       t211 = 4.0*t90*t11;
668       t212 = t103*e;
669       t214 = 2.0*t212*t54;
670       t215 = t53*x2;
671       t216 = 2.0*t215;
672       t217 = e*t11;
673       t218 = 2.0*t217;
674       t219 = 8.0*t171;
675       t220 = t108*x1;
676       t221 = 2.0*t220;
677       t222 = t8*t36;
678       t224 = t8*t34;
679       t225 = 12.0*t224;
680       t227 = (t216-t218+t219-t221-4.0*t222+t225)*x4;
681       t228 = 4.0*t113*t18;
682       t229 = 4.0*t57*t36;
683       t230 = t8*t29;
684       t231 = 2.0*t230;
685       t232 = t210+t211-t214+t227+t228+t229+t200+t231;
686       t234 = 4.0*t90*x2;
```

```
687       t235 = 6.0*t106;
688       t236 = 4.0*t113*x4;
689       t237 = 6.0*t115;
690       t238 = 4.0*t57*z;
691       t239 = t234-t235+t236+t105-t110+t114-t237-t119-t238;
692       t241 = t198+t200;
693       t242 = t241*x2;
694       t243 = t103*f;
695       t244 = t243*t28;
696       t245 = t199+t230;
697       t246 = 2.0*t245*x4;
698       t247 = 4.0*t57*f;
699       t248 = t247*t28;
700       t249 = t123*t1;
701       t252 = 2.0*t8*t125*t1;
702       t253 = t201*x2+t208*t11+t232*x4+t239*t18+t242-t244+t246-t248-t249+
703   ↪ t252-t63+t2;
704       t255 = t134*x3;
705       t256 = 2.0*t255;
706       t258 = (-4.0*t177+t180+t184-t186+t188-t256)*x2;
707       t259 = 4.0*t57*t22;
708       t260 = t8*t22;
709       t261 = 2.0*t260;
710       t262 = 2.0*t222;
711       t263 = t134*x1;
712       t264 = 2.0*t263;
713       t265 = t216-t218-t261+t219+t225-t262-t264;
714       t266 = t265*x4;
715       t267 = t156*e;
716       t269 = 2.0*t267*t54;
717       t270 = t258+t211+t259+t266+t228-t269+t200+t231;
718       t272 = t234+t105-t235-t133+t114-t237-t136-t238+t236;
719       t274 = t265*x2;
720       t275 = 4.0*t113*t11;
721       t277 = t144*x4;
722       t279 = d*t18;
723       t281 = t71*t36;
724       t283 = t77*d;
725       t284 = t283*x3;
726       t285 = 6.0*t284;
727       t287 = (12.0*t187+6.0*t277-6.0*t279-12.0*t281-t285)*x4;
728       t288 = 12.0*t146*t18;
729       t289 = t156*t36;
730       t290 = t71*t29;
731       t291 = 6.0*t290;
732       t292 = t274+t275-t196+t287+t288+t289+t231+t291;
733       t299 = 4.0*t113*x2-6.0*t141+12.0*t146*x4+t140+t97+t147-18.0*t148+
```

```
734 ↪ t150-t156*z;
735       t301 = 2.0*t245*x2;
736       t302 = t231+t291;
737       t303 = t302*x4;
738       t304 = t156*f;
739       t305 = t304*t28;
740       t306 = t158*t1;
741       t307 = t270*x2+t272*t11+t292*x4+t299*t18+t301-t248+t303-t305+t252-
742 ↪ t306-t63+t2;
743       t308 = b*t22;
744       t310 = b*b;
745       t311 = t310*t5;
746       t313 = 6.0*t167;
747       t316 = (-18.0*t308+108.0*t311-t313+12.0*t170)*x2;
748       t317 = t81*t11;
749       t320 = 24.0*t83*t22;
750       t321 = t103*b;
751       t323 = 6.0*t321*x1;
752       t324 = t44*t40;
753       t326 = t94*x1;
754       t327 = t326*x3;
755       t328 = 24.0*t327;
756       t329 = 48.0*t324+t328;
757       t330 = t329*x4;
758       t331 = 4.0*t90*e;
759       t332 = t331*t54;
760       t333 = 4.0*t332;
761       t335 = 2.0*t194*x3;
762       t336 = b*f;
763       t337 = t336*t28;
764       t340 = (t316+12.0*t317+t320-t323+t330-t333-t335+6.0*t337)*x2;
765       t345 = 12.0*t83*z;
766       t346 = 18.0*t161-18.0*t163-24.0*t165-t169+20.0*t171-t345;
767       t347 = t346*t11;
768       t348 = t208*t22;
769       t349 = t329*x2;
770       t350 = 4.0*t90*t22;
771       t351 = 12.0*t83*e;
772       t353 = 2.0*t351*t54;
773       t355 = 2.0*t212*x3;
774       t356 = e*t22;
775       t357 = 2.0*t356;
776       t358 = 8.0*t170;
777       t359 = t19*e;
778       t360 = 12.0*t359;
779       t361 = 2.0*t108;
780       t362 = e*t36;
```

```
781       t364 = 8.0*t95;
782       t365 = t21*d;
783       t366 = 12.0*t365;
784       t367 = -t357+t358+t360-t361-4.0*t362+t364+t366;
785       t368 = t367*x4;
786       t369 = t53*t18;
787       t370 = 4.0*t369;
788       t371 = 4.0*t113*e;
789       t372 = t371*t54;
790       t373 = 2.0*t372;
791       t374 = 4.0*t90*t36;
792       t376 = 2.0*t194*x1;
793       t377 = e*f;
794       t378 = t377*t28;
795       t379 = 2.0*t378;
796       t380 = t349+t350-t353-t355+t368+t370-t373+t374-t376+t379;
797       t381 = t380*x4;
798       t383 = 6.0*t183;
799       t384 = 6.0*t185;
800       t385 = 16.0*t187;
801       t386 = 4.0*t90*z;
802       t387 = -6.0*t177+t383+t180-t182-t384+t385-t190-t386;
803       t388 = t387*t18;
804       t389 = t239*e;
805       t391 = 2.0*t389*t54;
806       t392 = t28*x2;
807       t393 = t336*t392;
808       t394 = 6.0*t393;
809       t395 = 12.0*t83*f;
810       t396 = t395*t28;
811       t397 = t28*x4;
812       t398 = t377*t397;
813       t399 = 2.0*t398;
814       t400 = 4.0*t90*f;
815       t401 = t400*t28;
816       t402 = t40*t2;
817       t403 = 6.0*t402;
818       t404 = t44*t2;
819       t405 = 2.0*t404;
820       t406 = t340+t347+t348+t381+t388-t391+t394-t396+t399-t401+t403+t405;
821       t410 = 8.0*t199;
822       t413 = -36.0*t80+48.0*t82;
823       t418 = -t178+36.0*t179-t182+t383-t384+t188-6.0*t189-t386;
824       t422 = -12.0*t89+16.0*t88;
825       t424 = t346*x2+t174+t175+t176+t192+t193-t196+24.0*t197+t410+t413*t11-
826   ↪ t208*z+t418*x4+t422*t18;
827       t426 = t380*x2;
```

```
828       t427 = t418*t11;
829       t428 = t208*e;
830       t430 = 2.0*t428*t54;
831       t431 = t367*x2;
832       t432 = t53*t11;
833       t433 = 4.0*t432;
834       t435 = 2.0*t212*x1;
835       t436 = t8*t71;
836       t438 = t328+48.0*t436;
837       t439 = t438*x4;
838       t440 = 4.0*t113*t36;
839       t441 = 2.0*t440;
840       t442 = 4.0*t57*d;
841       t444 = 6.0*t442*x3;
842       t446 = (t431+t433-t333-t435+t439+t441-t444+t379)*x4;
843       t447 = 6.0*t215;
844       t448 = 6.0*t217;
845       t449 = 16.0*t171;
846       t450 = 8.0*t222;
847       t451 = 4.0*t113*z;
848       t452 = t447-t448+t449-t221-t450+t225-t451;
849       t453 = t452*t18;
850       t454 = t239*t36;
851       t455 = t377*t392;
852       t456 = 2.0*t455;
853       t457 = 4.0*t113*f;
854       t458 = t457*t28;
855       t459 = t8*t2;
856       t460 = 2.0*t459;
857       t461 = t426+t427-t430+t446+t453+t454+t456-t401+t399-t458+t405+t460;
858       t464 = t422*t11;
859       t466 = 8.0*t230;
860       t469 = -12.0*t111+16.0*t112;
861       t470 = t469*t18;
862       t472 = t387*x2+t464+t452*x4+t210+t211-t214+t227+t228+t229+t410+t466+
863   ↪ t470-t239*z;
864       t475 = (t394-t396+t399-t401+t403+t405)*x2;
865       t478 = 18.0*t197+6.0*t199;
866       t479 = t478*t11;
867       t480 = t208*f;
868       t481 = t480*t28;
869       t482 = t456-t401+t399-t458+t405+t460;
870       t483 = t482*x4;
871       t484 = 6.0*t245*t18;
872       t485 = t239*f;
873       t486 = t485*t28;
874       t488 = (t403+t405)*x2;
```

```
875       t489 = t243*t1;
876       t490 = t404+t459;
877       t491 = 2.0*t490*x4;
878       t492 = t247*t1;
879       t493 = t406*x2+t424*t11+t461*x4+t472*t18+t475+t479-t481+t483+t484-
880   ↪ t486+t488-t489+t491-t492+t124-t128-t129-t29;
881       t494 = 2.0*t350;
882       t495 = 4.0*t57*b;
883       t497 = 6.0*t495*x1;
884       t499 = 2.0*t362;
885       t500 = 2.0*t134;
886       t501 = -4.0*t356+t360+t358+t366-t499+t364-t500;
887       t502 = t501*x4;
888       t503 = 4.0*t372;
889       t505 = 2.0*t267*x3;
890       t507 = (t349+t494-t497+t502+t370-t503-t505+t379)*x2;
891       t508 = 8.0*t177;
892       t509 = -t508+t180+t383-t384+t385-t256-t386;
893       t510 = t509*t11;
894       t511 = t272*t22;
895       t512 = t501*x2;
896       t513 = 4.0*t113*t22;
897       t514 = 2.0*t332;
898       t515 = 12.0*t146*e;
899       t517 = 2.0*t515*t54;
900       t519 = 2.0*t267*x1;
901       t520 = t512+t433+t513-t514-t335+t439-t517+t440-t519+t379;
902       t521 = t520*x4;
903       t524 = t447-t448-6.0*t260+t219+36.0*t224-t262-t264-t451;
904       t525 = t524*t18;
905       t526 = t299*e;
906       t528 = 2.0*t526*t54;
907       t529 = t507+t510+t511+t521+t525-t528+t456-t401+t399-t458+t405+t460;
908       t534 = t447-t448-t261+t449+t225-6.0*t222-t264-t451;
909       t536 = t509*x2+t258+t211+t259+t266+t228-t269+t410+t466+t464-t272*z+
910   ↪ t534*x4+t470;
911       t538 = t520*x2;
912       t539 = t534*t11;
913       t540 = t272*e;
914       t542 = 2.0*t540*t54;
915       t543 = t438*x2;
916       t545 = d*t36;
917       t547 = d*d;
918       t548 = t547*t9;
919       t550 = 6.0*t283;
920       t552 = (12.0*t95-18.0*t545+108.0*t548-t550)*x4;
921       t553 = t144*t18;
```

```
922       t556 = 24.0*t146*t36;
923       t557 = t156*d;
924       t559 = 6.0*t557*x3;
925       t560 = d*f;
926       t561 = t560*t28;
927       t564 = (t543-t503-t376+t552+12.0*t553+t556-t559+6.0*t561)*x4;
928       t569 = 12.0*t146*z;
929       t570 = 20.0*t187+18.0*t277-18.0*t279-24.0*t281-t285-t569;
930       t571 = t570*t18;
931       t572 = t299*t36;
932       t573 = t560*t397;
933       t574 = 6.0*t573;
934       t575 = 12.0*t146*f;
935       t576 = t575*t28;
936       t577 = t71*t2;
937       t578 = 6.0*t577;
938       t579 = t538+t539-t542+t564+t571+t572+t456-t458+t574-t576+t460+t578;
939       t587 = -36.0*t143+48.0*t145;
940       t590 = t524*x2+t469*t11+t570*x4+t274+t275-t196+t287+t288+t289+t466+
941   ↪ 24.0*t290+t587*t18-t299*z;
942       t592 = t482*x2;
943       t593 = 6.0*t245*t11;
944       t594 = t272*f;
945       t595 = t594*t28;
946       t597 = (t456-t458+t574-t576+t460+t578)*x4;
947       t600 = 6.0*t230+18.0*t290;
948       t601 = t600*t18;
949       t602 = t299*f;
950       t603 = t602*t28;
951       t604 = 2.0*t490*x2;
952       t606 = (t460+t578)*x4;
953       t607 = t304*t1;
954       t608 = t529*x2+t536*t11+t579*x4+t590*t18+t592+t593-t595+t597+t601-
955   ↪ t603+t604-t492+t606-t607-t128+t159-t129-t29;
956
957       U[0][0] = x2;
958       U[0][1] = t11;
959       U[0][2] = x4;
960       U[0][3] = t18;
961       U[0][4] = 1.0;
962
963       U[1][0] = t11;
964       U[1][1] = t30;
965       U[1][2] = t18;
966       U[1][3] = t39;
967       U[1][4] = 0.0;
968
```

```
969         U[2][0] = t30;
970         U[2][1] = t64;
971         U[2][2] = t39;
972         U[2][3] = t79;
973         U[2][4] = 0.0;
974
975         U[3][0] = t64;
976         U[3][1] = t130;
977         U[3][2] = t79;
978         U[3][3] = t160;
979         U[3][4] = 0.0;
980
981         U[4][0] = t130;
982         U[4][1] = t253;
983         U[4][2] = t160;
984         U[4][3] = t307;
985         U[4][4] = 0.0;
986
987         U[5][0] = t253;
988         U[5][1] = t493;
989         U[5][2] = t307;
990         U[5][3] = t608;
991         U[5][4] = 0.0;
992
993         NX[0] = x1;
994         NX[1] = x2;
995         NX[2] = x3;
996         NX[3] = x4;
997         NX[4] = x5;
998
999         for (i=0;i<SYSORDER;i++) {
1000            for (j=0;j<APXORDER;j++) {
1001                NX[i] += U[j][i]*h[j];
1002            }
1003        }
1004
1005        if(lyapswitch=1) {
1006            /* Maple optimizing variables for dU */
1007            t609 = t8*x3;
1008            t610 = 2.0*t609;
1009            t611 = t63-t2;
1010            t612 = -t124+t128+t129+t29;
1011            t613 = t128-t159+t129+t29;
1012            t614 = t242-t244+t246-t248-t249+t252-t63+t2;
1013            t615 = t301-t248+t303-t305+t252-t306-t63+t2;
1014            t616 = t475+t479-t481+t483+t484-t486+t488-t489+t491-t492+t124-
1015 ↪ t128-t129-t29;
```

```
1016          t617 = t592+t593-t595+t597+t601-t603+t604-t492+t606-t607-t128+
1017  ↪ t159-t129-t29;
1018          t630 = 24.0*t94*x3;
1019          t631 = 48.0*t44*b+t630;
1020          t633 = t331*x3;
1021          t642 = -42.0*t308+180.0*t311-t313+20.0*t170-12.0*t49*b;
1022          t649 = t631*x2;
1023          t651 = 4.0*t90*b*x1;
1024          t660 = 24.0*t326;
1025          t661 = 48.0*e*b*x1+t660;
1026          t662 = t661*x4;
1027          t664 = z*t94*t54;
1028          t665 = 16.0*t664;
1029          t666 = t371*x3;
1030          t668 = t331*x1;
1031          t670 = 2.0*t194;
1032          t671 = t649-6.0*t651-24.0*t81*t609-4.0*t351*x3+t662-t665-2.0*
1033  ↪ t666-4.0*t668-t670;
1034          t674 = 48.0*t327;
1035          t675 = 72.0*t324+t674;
1036          t683 = 12.0*t81*t29;
1037          t685 = 6.0*t336*t1;
1038          t694 = 40.0*t327;
1039          t695 = 96.0*t324+t694;
1040          t699 = 2.0*t422*e*t54;
1041          t700 = t642*x2+t316+60.0*t317+t320-t323+t330-t333-t335+24.0*t337+
1042  ↪ t413*t22-t346*z+t695*x4-t699;
1043          t711 = t661*x2;
1044          t712 = t53*t22;
1045          t714 = 4.0*t633;
1046          t721 = t630+48.0*e*d*x3;
1047          t722 = t721*x4;
1048          t723 = t53*t36;
1049          t725 = t371*x1;
1050          t726 = 4.0*t725;
1051          t730 = t711+4.0*t712-t714-2.0*t351*x1-2.0*t212+t722+8.0*t723-
1052  ↪ t726-24.0*t90*d*x3;
1053          t733 = 16.0*t170;
1054          t735 = 16.0*t95;
1055          t737 = 4.0*t49*e;
1056          t738 = -6.0*t356+t733+t360-t361-8.0*t362+t735+t366-t737;
1057          t747 = 4.0*t53*t29;
1058          t749 = 2.0*t377*t1;
1059          t750 = t671*x2+t695*t11+t418*t22-2.0*t346*e*t54-2.0*t428*x3+t730*
1060  ↪ x4+t738*t18-2.0*t452*e*t54+t387*t36-2.0*t389*x1-t747+t749;
1061          t752 = t675*x2;
1062          t753 = t422*t22;
```

```
1063          t755 = 20.0*t369;
1064          t756 = 8.0*t378;
1065          t759 = 2.0*t469*e*t54;
1066          t761 = t752+t753+t738*x4+t349+t350-t353-t355+t368+t755-t373+t374-
1067    ↪ t376+t756-t759-t387*z;
1068          t767 = (-t683+t685)*x2;
1069          t768 = t28*t11;
1070          t770 = 18.0*t336*t768;
1071          t771 = t478*t22;
1072          t773 = t346*f*t28;
1073          t774 = -t747+t749;
1074          t775 = t774*x4;
1075          t776 = t28*t18;
1076          t778 = 6.0*t377*t776;
1077          t781 = 12.0*t245*e*t54;
1078          t783 = t387*f*t28;
1079          t784 = t1*x2;
1080          t786 = 6.0*t336*t784;
1081          t787 = t395*t1;
1082          t788 = t1*x4;
1083          t790 = 2.0*t377*t788;
1084          t791 = t400*t1;
1085          t792 = ((360.0*t310*x1*x2+36.0*t81*t22-216.0*t83*b*x1-6.0*t321+
1086    ↪ t631*x4-6.0*t633)*x2+t642*t11+2.0*t346*t22-6.0*t208*b*x1+t671*x4+t675*
1087    ↪ t18-4.0*t387*e*t54-2.0*t389*x3-t683+t685)*x2+t700*t11+t424*t22+t750*x4+
1088    ↪ t761*t18-2.0*t472*e*t54+t767+t770+t771-t773+t775+t778-t781-t783+t786-
1089    ↪ t787+t790-t791-t198-t200;
1090          t794 = 2.0*t401;
1091          t796 = 10.0*t398;
1092          t798 = 10.0*t404;
1093          t799 = t478*z;
1094          t800 = 28.0*t183;
1095          t803 = 24.0*t185;
1096          t806 = 2.0*t386;
1097          t808 = 2.0*t422*z;
1098          t809 = -t508+t800+48.0*t179-4.0*t181-t803+24.0*t187-8.0*t189-
1099    ↪ t806-t808;
1100          t812 = t413*f*t28;
1101          t814 = t422*f*t28;
1102          t832 = 36.0*t359;
1103          t834 = 36.0*t365;
1104          t835 = -t357+t358+t832-t361-12.0*t362+t364+t834-t737;
1105          t838 = t695*x2+t349+t350-t353-t355+t368+t755-t373+t374-t376+t756-
1106    ↪ t418*z-2.0*t413*e*t54+t835*x4+t422*t36;
1107          t840 = -2.0*t396-t794-t391+t348+t347+t381+t388+t340+30.0*t393+
1108    ↪ t796+30.0*t402+t798-t799+t809*t18-t812-t814+t700*x2+(84.0*t161-72.0*t163-
1109    ↪ 48.0*t165-12.0*t168+40.0*t171-2.0*t345-2.0*t413*z)*t11-t424*z+t838*x4;
```

```
1110          t853 = t721*x2;
1111          t854 = 24.0*t664;
1112          t858 = t660+48.0*t8*d;
1113          t859 = t858*x4;
1114          t861 = 4.0*t113*d*x3;
1115          t867 = t674+72.0*t436;
1116          t877 = 20.0*t432;
1117          t878 = t867*x4;
1118          t879 = t469*t36;
1119          t881 = t738*x2+t877-t699+t878+t431-t333-t435+t439+t441-t444+t756+
1120  ↪ t879-t452*z;
1121          t884 = t774*x2;
1122          t886 = 6.0*t377*t768;
1123          t889 = 2.0*t478*e*t54;
1124          t891 = t418*f*t28;
1125          t892 = 6.0*t245*t36;
1126          t894 = t452*f*t28;
1127          t896 = 2.0*t377*t784;
1128          t897 = t457*t1;
1129          t898 = t750*x2+t838*t11-2.0*t424*e*t54+(t730*x2+t835*t11-4.0*
1130  ↪ t418*e*t54-2.0*t428*x1+(t853-t854-6.0*t668+t859-18.0*t861-6.0*t442)*x4+
1131  ↪ t867*t18+2.0*t452*t36-6.0*t239*d*x3-t747+t749)*x4+t881*t18+t472*t36+t884+
1132  ↪ t886-t889-t891+t775+t778+t892-t894+t896-t791+t790-t897-t200-t231;
1133          t899 = 10.0*t459;
1134          t900 = 10.0*t455;
1135          t903 = t469*f*t28;
1136          t906 = 28.0*t215;
1137          t907 = 24.0*t217;
1138          t912 = 2.0*t451;
1139          t914 = 2.0*t469*z;
1140          t918 = 6.0*t245*z;
1141          t919 = 2.0*t458;
1142          t920 = -t794+t454+t899+t796-t430+t900+t798+t426+t427+t446+t453+
1143  ↪ t761*x2-t814-t903+t809*t11+t881*x4+(t906-t907+32.0*t171-4.0*t220-16.0*
1144  ↪ t222+24.0*t224-t912-t914)*t18-t472*z-t918-t919;
1145          t921 = t767+t770-t773+t771+t775+t778-t783-t781+t786-t787+t790-
1146  ↪ t791-t198-t200;
1147          t924 = 8.0*t398;
1148          t926 = 8.0*t404;
1149          t931 = t884+t886-t891-t889+t775+t778-t894+t892+t896-t791+t790-
1150  ↪ t897-t200-t231;
1151          t933 = 8.0*t455;
1152          t934 = 8.0*t459;
1153          t935 = t933-t814+t924-t401-t458+t926+t934-t903-t918;
1154          t949 = t896-t791+t790-t897-t200-t231;
1155          t951 = t921*x2+(24.0*t393-t396+t924-t401+24.0*t402+t926-t812-
1156  ↪ t799-t814)*t11-t424*f*t28+t931*x4+t935*t18-t472*f*t28+(t786-t787+t790-
```

```
1157 ↪ t791-t198-t200)*x2+(18.0*t402+6.0*t404)*t11-2.0*t478*f*t28-t480*t1+
1158 ↪ t949*x4;
1159         t955 = 12.0*t245*f*t28;
1160         t959 = 6.0*t490*t18-t955-t485*t1-t241*x2+t244-2.0*t245*x4+t248+
1161 ↪ t249-t252+t63-t2;
1162         t980 = t711+8.0*t712-24.0*t113*b*x1-t714+t722-2.0*t515*x3+4.0*
1163 ↪ t723-t726-2.0*t267;
1164         t983 = -12.0*t356+t832+t358+t834-t499+t364-t500-t737;
1165         t995 = -8.0*t356+t360+t733+t366-6.0*t362+t735-t500-t737;
1166         t997 = t752+t349+t494-t497+t502+t755-t503-t505+t756+t753-t509*z+
1167 ↪ t995*x4-t759;
1168         t1015 = t853-t665-4.0*t666-2.0*t668-t670+t859-24.0*t144*t609-4.0*
1169 ↪ t515*x1-6.0*t861;
1170         t1018 = t694+96.0*t436;
1171         t1026 = t980*x2+t995*t11+t534*t22-2.0*t509*e*t54-2.0*t540*x3+
1172 ↪ t1015*x4+t1018*t18-2.0*t570*e*t54+t524*t36-2.0*t526*x1-t747+t749;
1173         t1035 = t983*x2+t877+t469*t22+t1018*x4+t512+t513-t514-t335+t439-
1174 ↪ t517+t440-t519+t756-2.0*t587*e*t54-t524*z;
1175         t1040 = 6.0*t245*t22;
1176         t1042 = t509*f*t28;
1177         t1045 = 2.0*t600*e*t54;
1178         t1047 = t524*f*t28;
1179         t1048 = ((t649-18.0*t651-6.0*t495+t662-t854-6.0*t666)*x2+t675*
1180 ↪ t11+2.0*t509*t22-6.0*t272*b*x1+t980*x4+t983*t18-4.0*t524*e*t54-2.0*t526*
1181 ↪ x3-t747+t749)*x2+t997*t11+t536*t22+t1026*x4+t1035*t18-2.0*t590*e*t54+
1182 ↪ t884+t886+t1040-t1042+t775+t778-t1045-t1047+t896-t791+t790-t897-t200-
1183 ↪ t231;
1184         t1059 = t995*x2+t512+t877+t513-t514-t335+t439-t517+t440-t519+
1185 ↪ t756-t534*z-t699+t878+t879;
1186         t1065 = t906-t907-8.0*t260+24.0*t171+48.0*t224-t450-4.0*t263-
1187 ↪ t912-t914;
1188         t1067 = -t794+t899+t507+t510+t521+t511+t525-t528+t796+t900+t798-
1189 ↪ t814+t997*x2+(-16.0*t177+24.0*t179+t800-t803+32.0*t187-4.0*t255-t806-
1190 ↪ t808)*t11-t536*z+t1059*x4+t1065*t18-t903-t918-t919;
1191         t1098 = 20.0*t95-42.0*t545+180.0*t548-t550-12.0*t49*d;
1192         t1106 = 12.0*t144*t29;
1193         t1108 = 6.0*t560*t1;
1194         t1117 = t1018*x2-t759+t1098*x4+t543-t503-t376+t552+60.0*t553+
1195 ↪ t556-t559+24.0*t561+t587*t36-t570*z;
1196         t1121 = t534*f*t28;
1197         t1123 = (-t1106+t1108)*x4;
1198         t1125 = 18.0*t560*t776;
1199         t1126 = t600*t36;
1200         t1128 = t570*f*t28;
1201         t1130 = 6.0*t560*t788;
1202         t1131 = t575*t1;
1203         t1132 = t1026*x2+t1059*t11-2.0*t536*e*t54+(t1015*x2+t867*t11-4.0*
```

```
1204 ↪ t534*e*t54-2.0*t540*x1+(t858*x2-6.0*t725+360.0*t547*x3*x4+36.0*t144*t36-
1205 ↪ 216.0*t146*d*x3-6.0*t557)*x4+t1098*t18+2.0*t570*t36-6.0*t299*d*x3-t1106+
1206 ↪ t1108)*x4+t1117*t18+t590*t36+t884+t886-t781-t1121+t1123+t1125+t1126-
1207 ↪ t1128+t896-t897+t1130-t1131-t231-t291;
1208           t1149 = t600*z;
1209           t1151 = t587*f*t28;
1210           t1153 = t899-2.0*t576+t538+t539+t564+t571-t542+t572+30.0*t573+
1211 ↪ 30.0*t577+t900-t903+t1035*x2+t1065*t11+(84.0*t277+40.0*t187-72.0*t279-
1212 ↪ 48.0*t281-12.0*t284-2.0*t569-2.0*t587*z)*t18-t590*z-t1149-t1151+t1117*
1213 ↪ x4-t919;
1214           t1154 = t884+t886-t1042+t1040+t775+t778-t1047-t1045+t896-t791+
1215 ↪ t790-t897-t200-t231;
1216           t1159 = t884+t886-t1121-t781+t1123+t1125-t1128+t1126+t896-t897+
1217 ↪ t1130-t1131-t231-t291;
1218           t1172 = t1154*x2+t935*t11-t536*f*t28+t1159*x4+(t933-t903+24.0*
1219 ↪ t573-t458-t576+t934+24.0*t577-t1151-t1149)*t18-t590*f*t28+t949*x2+6.0*
1220 ↪ t490*t11-t955-t594*t1+(t896-t897+t1130-t1131-t231-t291)*x4;
1221           t1183 = (6.0*t459+18.0*t577)*t18-2.0*t600*f*t28-t602*t1-2.0*t245*
1222 ↪ x2+t248-t302*x4+t305-t252+t306+t63-t2;
1223
1224           dU[0][0][0] = 0.0;
1225           dU[0][0][1] = 1.0;
1226           dU[0][0][2] = 0.0;
1227           dU[0][0][3] = 0.0;
1228           dU[0][0][4] = 0.0;
1229           dU[0][1][0] = t22;
1230           dU[0][1][1] = -z;
1231           dU[0][1][2] = -t610;
1232           dU[0][1][3] = 0.0;
1233           dU[0][1][4] = -t29;
1234           dU[0][2][0] = 0.0;
1235           dU[0][2][1] = 0.0;
1236           dU[0][2][2] = 0.0;
1237           dU[0][2][3] = 1.0;
1238           dU[0][2][4] = 0.0;
1239           dU[0][3][0] = -t610;
1240           dU[0][3][1] = 0.0;
1241           dU[0][3][2] = t36;
1242           dU[0][3][3] = -z;
1243           dU[0][3][4] = -t29;
1244           dU[0][4][0] = 0.0;
1245           dU[0][4][1] = 0.0;
1246           dU[0][4][2] = 0.0;
1247           dU[0][4][3] = 0.0;
1248           dU[0][4][4] = 0.0;
1249
1250           dU[1][0][0] = t22;
```

```
1251            dU[1][0][1] = -z;
1252            dU[1][0][2] = -t610;
1253            dU[1][0][3] = 0.0;
1254            dU[1][0][4] = -t29;
1255            dU[1][1][0] = t47;
1256            dU[1][1][1] = t50;
1257            dU[1][1][2] = 2.0*t57;
1258            dU[1][1][3] = -t610;
1259            dU[1][1][4] = t611;
1260            dU[1][2][0] = -t610;
1261            dU[1][2][1] = 0.0;
1262            dU[1][2][2] = t36;
1263            dU[1][2][3] = -z;
1264            dU[1][2][4] = -t29;
1265            dU[1][3][0] = 2.0*t57;
1266            dU[1][3][1] = -t610;
1267            dU[1][3][2] = t75;
1268            dU[1][3][3] = t77;
1269            dU[1][3][4] = t611;
1270            dU[1][4][0] = 0.0;
1271            dU[1][4][1] = 0.0;
1272            dU[1][4][2] = 0.0;
1273            dU[1][4][3] = 0.0;
1274            dU[1][4][4] = 0.0;
1275
1276            dU[2][0][0] = t47;
1277            dU[2][0][1] = t50;
1278            dU[2][0][2] = 2.0*t57;
1279            dU[2][0][3] = -t610;
1280            dU[2][0][4] = t611;
1281            dU[2][1][0] = t98;
1282            dU[2][1][1] = t103;
1283            dU[2][1][2] = t120;
1284            dU[2][1][3] = 4.0*t57;
1285            dU[2][1][4] = t612;
1286            dU[2][2][0] = 2.0*t57;
1287            dU[2][2][1] = -t610;
1288            dU[2][2][2] = t75;
1289            dU[2][2][3] = t77;
1290            dU[2][2][4] = t611;
1291            dU[2][3][0] = t137;
1292            dU[2][3][1] = 4.0*t57;
1293            dU[2][3][2] = t151;
1294            dU[2][3][3] = t156;
1295            dU[2][3][4] = t613;
1296            dU[2][4][0] = 0.0;
1297            dU[2][4][1] = 0.0;
```

```
1298            dU[2][4][2] = 0.0;
1299            dU[2][4][3] = 0.0;
1300            dU[2][4][4] = 0.0;
1301
1302            dU[3][0][0] = t98;
1303            dU[3][0][1] = t103;
1304            dU[3][0][2] = t120;
1305            dU[3][0][3] = 4.0*t57;
1306            dU[3][0][4] = t612;
1307            dU[3][1][0] = t201;
1308            dU[3][1][1] = t208;
1309            dU[3][1][2] = t232;
1310            dU[3][1][3] = t239;
1311            dU[3][1][4] = t614;
1312            dU[3][2][0] = t137;
1313            dU[3][2][1] = 4.0*t57;
1314            dU[3][2][2] = t151;
1315            dU[3][2][3] = t156;
1316            dU[3][2][4] = t613;
1317            dU[3][3][0] = t270;
1318            dU[3][3][1] = t272;
1319            dU[3][3][2] = t292;
1320            dU[3][3][3] = t299;
1321            dU[3][3][4] = t615;
1322            dU[3][4][0] = 0.0;
1323            dU[3][4][1] = 0.0;
1324            dU[3][4][2] = 0.0;
1325            dU[3][4][3] = 0.0;
1326            dU[3][4][4] = 0.0;
1327
1328            dU[4][0][0] = t201;
1329            dU[4][0][1] = t208;
1330            dU[4][0][2] = t232;
1331            dU[4][0][3] = t239;
1332            dU[4][0][4] = t614;
1333            dU[4][1][0] = t406;
1334            dU[4][1][1] = t424;
1335            dU[4][1][2] = t461;
1336            dU[4][1][3] = t472;
1337            dU[4][1][4] = t616;
1338            dU[4][2][0] = t270;
1339            dU[4][2][1] = t272;
1340            dU[4][2][2] = t292;
1341            dU[4][2][3] = t299;
1342            dU[4][2][4] = t615;
1343            dU[4][3][0] = t529;
1344            dU[4][3][1] = t536;
```

```
1345            dU[4][3][2] = t579;
1346            dU[4][3][3] = t590;
1347            dU[4][3][4] = t617;
1348            dU[4][4][0] = 0.0;
1349            dU[4][4][1] = 0.0;
1350            dU[4][4][2] = 0.0;
1351            dU[4][4][3] = 0.0;
1352            dU[4][4][4] = 0.0;
1353
1354            dU[5][0][0] = t406;
1355            dU[5][0][1] = t424;
1356            dU[5][0][2] = t461;
1357            dU[5][0][3] = t472;
1358            dU[5][0][4] = t616;
1359            dU[5][1][0] = t792;
1360            dU[5][1][1] = t840;
1361            dU[5][1][2] = t898;
1362            dU[5][1][3] = t920;
1363            dU[5][1][4] = t951+t959;
1364            dU[5][2][0] = t529;
1365            dU[5][2][1] = t536;
1366            dU[5][2][2] = t579;
1367            dU[5][2][3] = t590;
1368            dU[5][2][4] = t617;
1369            dU[5][3][0] = t1048;
1370            dU[5][3][1] = t1067;
1371            dU[5][3][2] = t1132;
1372            dU[5][3][3] = t1153;
1373            dU[5][3][4] = t1172+t1183;
1374            dU[5][4][0] = 0.0;
1375            dU[5][4][1] = 0.0;
1376            dU[5][4][2] = 0.0;
1377            dU[5][4][3] = 0.0;
1378            dU[5][4][4] = 0.0;
1379
1380            for (i=0;i<APXORDER;i++) {
1381                for (j=0;j<SYSORDER;j++) {
1382                    for (k=0;k<SYSORDER;k++) {
1383                        DF[j][k] += dU[i][j][k]*h[i];
1384                    }
1385                }
1386            }
1387        }
1388    }
1389
1390    /*--------------------------------------------------------------------
1391     * Gram-Schmidt Orthonormalizing function (gramschimdt)
```

```
1392  *-------------------------------------------------------------------
1393  *  Purpose:
1394  *      Orthonormalizes the basis vectors of the state space based on
1395  *      Jacobian.  Returns vector magnitudes
1396  *
1397  *  Variables:
1398  *      Name:       Type:       Connotation:
1399  *      -----       -----       -----------
1400  *      V           double[][]  array of basis vectors (not orthonormalized)
1401  *      one         double      multiplier for BLAS function dgemm.f
1402  *      zero        double      addition value for BLAS function dgemm.f
1403  *      dumscale    double      scale factor for normalizing
1404  *      dumdot      double      -1 * dot product of 2 vectors
1405  *      vectptr     double *[]  array of pointers to rows(vectors) of V
1406  *      m           int         number of state equations
1407  *      i_one       int         increment of data for BLAS functions
1408  *      i           int         counter
1409  *      j           int         counter
1410  *      chn         char *      pointer to dgemm.f flag N (normal)
1411  *      cht         char *      pointer to dgemm.f flag T (transpose)
1412  *-------------------------------------------------------------------*/
1413 void gramschimidt(double DF[SYSORDER][SYSORDER],
1414                   double BMAT[SYSORDER][SYSORDER],double mags[SYSORDER])
1415 {
1416     double V[SYSORDER][SYSORDER]={0},one=1.0,zero=0.0,dumscale=0.0,
1417            dumdot=0.0,*vectptr[SYSORDER] = NULL;
1418     int    i,j,m=SYSORDER,i_one=1;
1419     char   *chn="n",*cht="t";
1420
1421     /* Apparent error in either the documentation or matlab
1422        implementation of dgemm.f
1423        Matrices A and B and associated flags needed to be swapped
1424        i.e. to perform A*B, must send B first and A second. */
1425     dgemm(cht,chn,&m,&m,&m,&one,*DF,&m,*BMAT,&m,&zero,*V,&m);
1426
1427     /* Construct pointers to individual row vectors from V
1428        This only works because the number of columns (m) is sent to the
1429        BLAS functions */
1430     for (i=0;i<SYSORDER;i++) {
1431         vectptr[i] = &V[i][0]; }
1432
1433     /* get norm of vect1 */
1434     mags[0] = dnrm2(&m,vectptr[0],&i_one);
1435     /* normalize vect1 */
1436     dumscale = 1.0/mags[0];
1437     dscal(&m,&dumscale,vectptr[0],&i_one);
1438
```

```
1439        for (i=1;i<SYSORDER;i++) {
1440            for (j=(i-1);j>-1;j--) {
1441                /* dot product of vect[i-j] and vect[i] multiplied by -1 for
1442                   daxpy function */
1443                dumdot = -1*ddot(&m,vectptr[i],&i_one,vectptr[j],&i_one);
1444                /* vect[i] - (vect[i] dot vect[j])*vect[j] */
1445                daxpy(&m,&dumdot,vectptr[j],&i_one,vectptr[i],&i_one);
1446            }
1447            mags[i] = dnrm2(&m,vectptr[i],&i_one);
1448            dumscale = 1.0/mags[i];
1449            dscal(&m,&dumscale,vectptr[i],&i_one);
1450        }
1451
1452        /* Reset basis matrix */
1453        for (i=0;i<SYSORDER;i++) {
1454            for (j=0;j<SYSORDER;j++) {
1455                BMAT[i][j] = *(vectptr[i]+j);
1456            }
1457        }
1458    }
```

B.2 Poincaré Maps

```
/*-----------------------------------------------------------------------
 * Poincare mapping mex file (poincare4mex.c)
 *  Resamples time series data based on how many data points per cycle
 *  for a 4th order non-autonomous system
 *-----------------------------------------------------------------------
 *  Input Parameters:
 *      Name:       Type:       Connotation:
 *      -----       -----       ------------
 *      SIMFILE     mxArray*    Matlab string for sim datafile name
 *      POINFILE    mxArray*    Matlab string for poincare datafile name
 *
 *  Output parameters:
 *      Name:       Type:       Connotation:
 *      -----       -----       ------------
 *      TP_OUT      mxArray*    Poincare map variable (time)
 *      X1P_OUT     mxArray*    Poincare map variable (x1p)
 *      X2P_OUT     mxArray*    Poincare map variable (x2p)
 *      X3P_OUT     mxArray*    Poincare map variable (x3p)
 *      X4P_OUT     mxArray*    Poincare map variable (x4p)
 *      PERIODS     mxArray*    number of periods of data in sim file
 *
 *  Variables:
 *      Name:       Type:       Connotation:
 *      -----       -----       ------------
 *      mxFILE      mxArray*    Matlab filename string
 *      buflen      int         length of string buffer for file name
 *      simfile     char*       C filename string
 *      poinfile    char*       C filename string
 *      status      int         function call status variable
 *      mxPOINNAME  mxArray*    filename for storage in data file
 *      simmat      MATfile*    mat-file pointer
 *      poinmat     MATfile*    mat-file pointer
 *      mxT         mxArray*    t values loaded from file
 *      mxX1        mxArray*    x1 values loaded from file
 *      mxX2        mxArray*    x2 values loaded from file
 *      mxX3        mxArray*    x3 values loaded from file
 *      mxX4        mxArray*    x4 values loaded from file
 *      t           double*     pointer to mxT
 *      x1          double*     pointer to mxX1
 *      x2          double*     pointer to mxX2
 *      x3          double*     pointer to mxX3
 *      x4          double*     pointer to mxX4
 *      steps       int         number of time steps
 *      rawcdiv     double      number of data points per cycles (error
 *                                  prone, should be an integer)
 *      rem         double      non-rational portion of rawcdiv
```

```
47  *       cdiv        int         number of data points per cycle
48  *       points      int         number of Poincare map points
49  *       tp          double*     pointer to TP_OUT
50  *       x1p         double*     pointer to X1P_OUT
51  *       x2p         double*     pointer to X2P_OUT
52  *       x3p         double*     pointer to X3P_OUT
53  *       x4p         double*     pointer to X4P_OUT
54  *       i           int         counter
55  *-------------------------------------------------------------------*/
56  #include "mex.h"
57  #include "mat.h"
58  #include <stdio.h>
59  #include <math.h>
60
61  #define SIMFILE     prhs[0]
62  #define POINFILE    prhs[1]
63  #define TP_OUT      plhs[0]
64  #define X1P_OUT     plhs[1]
65  #define X2P_OUT     plhs[2]
66  #define X3P_OUT     plhs[3]
67  #define X4P_OUT     plhs[4]
68  #define PERIODS     plhs[5]
69
70  #define PI          3.1415926535898
71
72  void mexFunction( int nlhs, mxArray *plhs[],
73                    int nrhs, const mxArray *prhs[] )
74  {
75      int     buflen,status,cdiv,steps,points,i;
76      char    *simfile,*poinfile;
77      mxArray *mxT,*mxX1,*mxX2,*mxX3,*mxX4,*mxPOINNAME;
78      double  rawcdiv,rem,*t,*x1,*x2,*x3,*x4,*tp,*x1p,*x2p,*x3p,*x4p;
79      MATFile *simmat,*poinmat;
80
81      /* check for input. */
82      if(nrhs != 2)
83          mexErrMsgTxt("\n Invalid inputs \n\trequired: sim filename, poincare
84  ↪ filename");
85
86      /* Get filenames */
87      buflen = (mxGetM(SIMFILE) * mxGetN(SIMFILE) * sizeof(mxChar)) + 1;
88      simfile = mxCalloc(buflen, sizeof(char));
89      status = mxGetString(SIMFILE, simfile, buflen);
90      if(status != 0)
91          mexErrMsgTxt("SIM filename error.");
92
93      buflen = (mxGetM(POINFILE) * mxGetN(POINFILE) * sizeof(mxChar)) + 1;
```

```
94      poinfile = mxCalloc(buflen, sizeof(char));
95      status = mxGetString(POINFILE, poinfile, buflen);
96      if(status != 0)
97          mexErrMsgTxt("POINCARE filename error.");
98      mxPOINNAME = mxCreateString(poinfile);
99
100     printf("\n Reading sim file:      %s ...", simfile);
101
102     /* Open sim data file */
103     simmat = matOpen(simfile, "r");
104     if(simmat == NULL) {
105         printf("\n Error opening file: %s", simfile);
106         return; }
107
108     /* Read in each array and reference */
109     mxT  = matGetVariable(simmat, "t");
110     mxX1 = matGetVariable(simmat, "x1");
111     mxX2 = matGetVariable(simmat, "x2");
112     mxX3 = matGetVariable(simmat, "x3");
113     mxX4 = matGetVariable(simmat, "x4");
114     t  = mxGetPr(mxT);
115     x1 = mxGetPr(mxX1);
116     x2 = mxGetPr(mxX2);
117     x3 = mxGetPr(mxX3);
118     x4 = mxGetPr(mxX4);
119     steps = mxGetM(mxT);
120
121     /* Calculate number of points per drive cycle */
122     rawcdiv = 2*PI/(t[1]-t[0]);
123     rem = fmod(rawcdiv,1.0);
124     if(rem<0.5)
125         cdiv = floor(rawcdiv);
126     else
127         cdiv = ceil(rawcdiv);
128     points = (steps-1)/cdiv+1;
129     PERIODS = mxCreateDoubleScalar(points-1);
130
131     /* Create and reference to output arrays */
132     TP_OUT  = mxCreateDoubleMatrix(points,1,mxREAL);
133     X1P_OUT = mxCreateDoubleMatrix(points,1,mxREAL);
134     X2P_OUT = mxCreateDoubleMatrix(points,1,mxREAL);
135     X3P_OUT = mxCreateDoubleMatrix(points,1,mxREAL);
136     X4P_OUT = mxCreateDoubleMatrix(points,1,mxREAL);
137     tp  = mxGetPr(TP_OUT);
138     x1p = mxGetPr(X1P_OUT);
139     x2p = mxGetPr(X2P_OUT);
140     x3p = mxGetPr(X3P_OUT);
```

```
141     x4p = mxGetPr(X4P_OUT);
142
143     /* Resample for mapping */
144     for (i=0;i<points;i++) {
145         tp[i]  = t[i*cdiv];
146         x1p[i] = x1[i*cdiv];
147         x2p[i] = x2[i*cdiv];
148         x3p[i] = x3[i*cdiv];
149         x4p[i] = x4[i*cdiv];
150     }
151
152     /* Destroy allocated memory */
153     mxDestroyArray(mxT);
154     mxDestroyArray(mxX1);
155     mxDestroyArray(mxX2);
156     mxDestroyArray(mxX3);
157     mxDestroyArray(mxX4);
158
159     /* Close sim data file */
160     if(matClose(simmat) != 0) {
161         printf("\n Error closing file: %s",simfile);
162         return; }
163
164     printf("\n Writing poincare file: %s ...", poinfile);
165
166     /* Open poincare data file */
167     poinmat = matOpen(poinfile, "w");
168     if(poinmat == NULL) {
169         printf("\n Error opening file: %s", poinfile);
170         return; }
171
172     /* Write poincare data file */
173     status = matPutVariable(poinmat, "tp", TP_OUT);
174     if(status != 0) {
175         printf("\n Error writing tp to %s",poinfile);
176         return; }
177     status = matPutVariable(poinmat, "x1p", X1P_OUT);
178     if(status != 0) {
179         printf("\n Error writing x1p to %s",poinfile);
180         return; }
181     status = matPutVariable(poinmat, "x2p", X2P_OUT);
182     if(status != 0) {
183         printf("\n Error writing x2p to %s",poinfile);
184         return; }
185     status = matPutVariable(poinmat, "x3p", X3P_OUT);
186     if(status != 0) {
187         printf("\n Error writing x3p to %s",poinfile);
```

```
188         return; }
189     status = matPutVariable(poinmat, "x4p", X4P_OUT);
190     if(status != 0) {
191         printf("\n Error writing x4p to %s",poinfile);
192         return; }
193     status = matPutVariable(poinmat, "datafile", mxPOINNAME);
194     if(status != 0) {
195         printf("\n Error writing datafile to %s",poinfile);
196         return; }
197     else { mxDestroyArray(mxPOINNAME); }
198
199     /* Close poincare data file */
200     if(matClose(poinmat) != 0) {
201         printf("\n Error closing file: %s",poinfile);
202         return; }
203
204     printf("\n\n Poincare Map created.");
205     printf("\n POINCARE4MEX.C TERMINATING ...\n");
206 }
```

B.3 Lyapunov Spectrum

```
/*-----------------------------------------------------------------------
 * Lyapunov spectrum diagram function (lyapspecmex.c)
 *  Loads final values of Lyapunov exponents from exponent data and
 *  creates spectrums
 *-----------------------------------------------------------------------
 *  Input Parameters:
 *      Name:       Type:       Connotation:
 *      -----       -----       ------------
 *      EXPFILES    mxArray*    Matlab cell array of exponent datafile names
 *
 *  Output parameters:
 *      Name:       Type:       Connotation:
 *      -----       -----       ------------
 *      SPEC1       mxArray*    array of final x1 Lyapunov exponents
 *      SPEC2       mxArray*    array of final x2 Lyapunov exponents
 *      SPEC3       mxArray*    array of final x3 Lyapunov exponents
 *      SPEC4       mxArray*    array of final x4 Lyapunov exponents
 *      SPEC5       mxArray*    array of final x5 Lyapunov exponents
 *      MINER       mxArray*    worst relative convergence of loaded data
 *      MINEA       mxArray*    worst absolute convergence of loaded data
 *
 *  Variables:
 *      Name:       Type:       Connotation:
 *      -----       -----       ------------
 *      size        int         number of file names in EXPFILES
 *      spec1       double*     pointer to SPEC1
 *      spec2       double*     pointer to SPEC2
 *      spec3       double*     pointer to SPEC3
 *      spec4       double*     pointer to SPEC4
 *      spec5       double*     pointer to SPEC5
 *      minEr       double*     pointer to MINER
 *      minEa       double*     pointer to MINEA
 *      mxFILE      mxArray*    Matlab filename string
 *      buflen      int         length of string buffer for file name
 *      filename    char*       C filename string
 *      status      int         function call status variable
 *      expmat      MATfile*    mat-file pointer
 *      mxFINALEXP1 mxArray*    finalexp1 values loaded from file
 *      mxFINALEXP2 mxArray*    finalexp2 values loaded from file
 *      mxFINALEXP3 mxArray*    finalexp3 values loaded from file
 *      mxFINALEXP4 mxArray*    finalexp4 values loaded from file
 *      mxFINALEXP5 mxArray*    finalexp5 values loaded from file
 *      mxER        mxArray*    Er value loaded from file
 *      mxEA        mxArray*    Ea value loaded from file
 *      finalexp1   double*     pointer to mxFINALEXP1
 *      finalexp2   double*     pointer to mxFINALEXP2
```

```
47  *       finalexp3   double*     pointer to mxFINALEXP3
48  *       finalexp4   double*     pointer to mxFINALEXP4
49  *       finalexp5   double*     pointer to mxFINALEXP5
50  *       Er          double*     pointer to mxER
51  *       Ea          double*     pointer to mxEA
52  *       i           int         counter
53  *-----------------------------------------------------------------------*/
54  #include "mex.h"
55  #include "mat.h"
56  #include <stdio.h>
57  #include <stdlib.h>
58
59  #define EXPFILES    prhs[0]
60  #define SPEC1       plhs[0]
61  #define SPEC2       plhs[1]
62  #define SPEC3       plhs[2]
63  #define SPEC4       plhs[3]
64  #define SPEC5       plhs[4]
65  #define MINER       plhs[5]
66  #define MINEA       plhs[6]
67
68  void mexFunction( int nlhs, mxArray *plhs[],
69                    int nrhs, const mxArray *prhs[] )
70  {
71      int     size,i,buflen,status;
72      double  *spec1,*spec2,*spec3,*spec4,*spec5,*minEr,*minEa,*finalexp1,
73              *finalexp2,*finalexp3,*finalexp4,*finalexp5,*Er,*Ea;
74      char    *filename;
75      MATFile *expmat;
76      mxArray *mxFILE,*mxFINALEXP1,*mxFINALEXP2,*mxFINALEXP3,*mxFINALEXP4,
77              *mxFINALEXP5,*mxER,*mxEA;
78
79      /* check for input. */
80      if(nrhs != 1)
81          mexErrMsgTxt("\n Input required: filename cell array");
82
83      size = mxGetNumberOfElements(EXPFILES);
84
85      /* Create and reference output arrays */
86      SPEC1 = mxCreateDoubleMatrix(size,1,mxREAL);
87      SPEC2 = mxCreateDoubleMatrix(size,1,mxREAL);
88      SPEC3 = mxCreateDoubleMatrix(size,1,mxREAL);
89      SPEC4 = mxCreateDoubleMatrix(size,1,mxREAL);
90      SPEC5 = mxCreateDoubleMatrix(size,1,mxREAL);
91      MINER = mxCreateDoubleMatrix(1,1,mxREAL);
92      MINEA = mxCreateDoubleMatrix(1,1,mxREAL);
93      spec1 = mxGetPr(SPEC1);
```

```
94      spec2 = mxGetPr(SPEC2);
95      spec3 = mxGetPr(SPEC3);
96      spec4 = mxGetPr(SPEC4);
97      spec5 = mxGetPr(SPEC5);
98      minEr = mxGetPr(MINER);
99      minEa = mxGetPr(MINEA);
100
101     /* Initialize minimum convergences */
102     *minEr = 0;
103     *minEa = 0;
104
105     /* Load each file and process */
106     for(i=0;i<size;i++) {
107         mxFILE = mxGetCell(EXPFILES,i);
108         buflen = (mxGetM(mxFILE) * mxGetN(mxFILE) * sizeof(mxChar)) + 1;
109         filename = mxCalloc(buflen, sizeof(char));
110         status = mxGetString(mxFILE, filename, buflen);
111         if(status != 0) {
112             mexErrMsgTxt("Exp filename error.");
113             return; }
114
115         printf("\nReading exp file: %s ...", filename);
116
117         /* Open exp data file */
118         expmat = matOpen(filename, "r");
119         if (expmat == NULL) {
120             printf("\nError opening file %s", filename);
121             return; }
122
123         /* Read in each array and reference */
124         mxFINALEXP1 = matGetVariable(expmat, "finalexp1");
125         mxFINALEXP2 = matGetVariable(expmat, "finalexp2");
126         mxFINALEXP3 = matGetVariable(expmat, "finalexp3");
127         mxFINALEXP4 = matGetVariable(expmat, "finalexp4");
128         mxFINALEXP5 = matGetVariable(expmat, "finalexp5");
129         mxER    = matGetVariable(expmat, "Er");
130         mxEA    = matGetVariable(expmat, "Ea");
131         finalexp1 = mxGetPr(mxFINALEXP1);
132         finalexp2 = mxGetPr(mxFINALEXP2);
133         finalexp3 = mxGetPr(mxFINALEXP3);
134         finalexp4 = mxGetPr(mxFINALEXP4);
135         finalexp5 = mxGetPr(mxFINALEXP5);
136         Er    = mxGetPr(mxER);
137         Ea    = mxGetPr(mxEA);
138
139         /* Get last value and add to output */
140         spec1[i] = *finalexp1;
```

```
141             spec2[i] = *finalexp2;
142             spec3[i] = *finalexp3;
143             spec4[i] = *finalexp4;
144             spec5[i] = *finalexp5;
145
146             /* Check for worst convergence */
147             if (*Er>*minEr) { *minEr = *Er; }
148             if (*Ea>*minEa) { *minEa = *Ea; }
149
150             /* Destroy allocated memory */
151             mxDestroyArray(mxFINALEXP1);
152             mxDestroyArray(mxFINALEXP2);
153             mxDestroyArray(mxFINALEXP3);
154             mxDestroyArray(mxFINALEXP4);
155             mxDestroyArray(mxFINALEXP5);
156             mxDestroyArray(mxER);
157             mxDestroyArray(mxEA);
158
159             /* Close exp data file */
160             if (matClose(expmat) != 0) {
161                 printf("\nError closing file %s",filename);
162                 return; }
163         }
164         printf("\n\nFinished generating Lyapunov exponent spectrums.");
165         printf("\nLYAPSPECMEX.C TERMINATING ...\n");
166     }
```

B.4 Bifurcation Diagrams

```
/*-----------------------------------------------------------------------
 * Bifurcation diagram function (bifurmex.c)
 *  Loads x1p and x3p from Poincare map data and creates two bifurcation
 *  diagrams
 *-----------------------------------------------------------------------
 *  Input Parameters:
 *      Name:       Type:       Connotation:
 *      -----       -----       ------------
 *      PARAMS      mxArray*    Matlab array of parameter values
 *      CIGNORE     mxArray*    Matlab number of cycles to ignore
 *      DATFILES    mxArray*    Matlab cell array of Poincare map datafile
 *                                  names
 *
 *  Output parameters:
 *      Name:       Type:       Connotation:
 *      -----       -----       ------------
 *      PERIODS     mxArray*    number of periods of data in files
 *
 *  Variables:
 *      Name:       Type:       Connotation:
 *      -----       -----       ------------
 *      params      double*     pointer to PARAMS
 *      c_ignore    double      number of cycles to ignore
 *      igcycles    int         number of cycles to ignore (as integer)
 *      size        int         number of file names in DATFILES
 *      mxX1B       mxArray*    all x1p values for bifurcation diagram
 *      mxX3B       mxArray*    all x3p values for bifurcation diagram
 *      mxPARAMB    mxArray*    all parameter values for bifurcation
 *                                  diagram
 *      x1b         double*     pointer to mxX1b
 *      x3b         double*     pointer to mxX3b
 *      paramb      double*     pointer to PARAMB
 *      mxFILE      mxArray*    Matlab filename string
 *      buflen      int         length of string buffer for file name
 *      filename    char*       C filename string
 *      status      int         function call status variable
 *      filemat     MATfile*    mat-file pointer
 *      mxX1P       mxArray*    x1p values loaded from file
 *      mxX3P       mxArray*    x3p values loaded from file
 *      x1p         double*     pointer to mxX1P
 *      x3p         double*     pointer to mxX3P
 *      cycles      int         number of cycles loaded from files
 *      xlhs        mxArray*    arguments for Matlab plot function
 *      ylhs        mxArray*    arguments for Matlab plot function
 *      i           int         counter
 *      j           int         counter
```

```
47   *-----------------------------------------------------------------*/
48  #include "mex.h"
49  #include "mat.h"
50  #include <stdio.h>
51  #include <stdlib.h>
52  #include <math.h>
53
54  #define PARAMS      prhs[0]
55  #define CIGNORE     prhs[1]
56  #define DATFILES    prhs[2]
57  #define PERIODS     plhs[0]
58  #define MAXCYCLES   3001
59
60  void mexFunction( int nlhs, mxArray *plhs[],
61                    int nrhs, const mxArray *prhs[] )
62  {
63      int     igcycles,size,i,j,buflen,status,cycles;
64      double  *params,c_ignore,*x1b,*x3b,*paramb,*x1p,*x3p;
65      char    *filename;
66      MATFile *filemat;
67      mxArray *mxX1B,*mxX3B,*mxPARAMB,*mxFILE,*mxX1P,*mxX3P,*xlhs[5],*ylhs[5];
68
69      /* Check for input. */
70      if(nrhs != 3)
71          mexErrMsgTxt("\n Invalid inputs \n\trequired: array of parameter
72  ↪ values, number of cycles to ignore, filename cell array");
73
74      /* Get inputs */
75      params = mxGetPr(PARAMS);
76      c_ignore = mxGetScalar(CIGNORE);
77      /* Convert to integer for use in array indexing */
78      igcycles = ceil(c_ignore);
79
80      size = mxGetNumberOfElements(DATFILES);
81      if(size != mxGetN(PARAMS))
82          mexErrMsgTxt("\n Improper number of filenames or parameters");
83
84      /* Create and reference to bifurcation diagram data */
85      mxX1B    = mxCreateDoubleMatrix((MAXCYCLES-igcycles)*size,1,mxREAL);
86      mxX3B    = mxCreateDoubleMatrix((MAXCYCLES-igcycles)*size,1,mxREAL);
87      mxPARAMB = mxCreateDoubleMatrix((MAXCYCLES-igcycles)*size,1,mxREAL);
88      x1b    = mxGetPr(mxX1B);
89      x3b    = mxGetPr(mxX3B);
90      paramb = mxGetPr(mxPARAMB);
91
92      /* Load each file and process */
93      for(i=0;i<size;i++) {
```

```
94          mxFILE = mxGetCell(DATFILES,i);
95          buflen = (mxGetM(mxFILE) * mxGetN(mxFILE) * sizeof(mxChar)) + 1;
96          filename = mxCalloc(buflen, sizeof(char));
97          status = mxGetString(mxFILE, filename, buflen);
98          if(status != 0)
99              mexErrMsgTxt("Sim datafile name error.");
100
101         printf("\nReading poincare file %s ...", filename);
102
103         /* Open poincare data file */
104         filemat = matOpen(filename, "r");
105         if (filemat == NULL) {
106             printf("\nError opening file %s", filename);
107             return; }
108
109         /* Read in each array and reference */
110         mxX1P = matGetVariable(filemat, "x1p");
111         mxX3P = matGetVariable(filemat, "x3p");
112         x1p = mxGetPr(mxX1P);
113         x3p = mxGetPr(mxX3P);
114         cycles = mxGetM(mxX1P);
115
116         /* Error check for ignored cycles */
117         if(cycles<c_ignore)
118             mexErrMsgTxt("Ignored cycles are greater than available data!");
119
120         /* Truncate to remove transient cycles to fill bifurcation data */
121         for(j=0;j<(cycles-igcycles);j++) {
122             x1b[j+(cycles-igcycles)*i]    = x1p[j+igcycles];
123             x3b[j+(cycles-igcycles)*i]    = x3p[j+igcycles];
124             paramb[j+(cycles-igcycles)*i] = params[i];
125         }
126
127         /* Destroy allocated memory */
128         mxDestroyArray(mxX1P);
129         mxDestroyArray(mxX3P);
130
131         /* Close poincare data file */
132         if (matClose(filemat) != 0) {
133             printf("\nError closing file %s",filename);
134             return; }
135     }
136     /* Get output */
137     PERIODS = mxCreateDoubleScalar(cycles-1);
138
139     /* Define plot arguments */
140     xlhs[0] = mxPARAMB;
```

```
141     xlhs[1] = mxX1B;
142     xlhs[2] = mxCreateString("b.");
143     xlhs[3] = mxCreateString("markersize");
144     xlhs[4] = mxCreateDoubleScalar(4);
145     ylhs[0] = mxPARAMB;
146     ylhs[1] = mxX3B;
147     ylhs[2] = mxCreateString("r.");
148     ylhs[3] = mxCreateString("markersize");
149     ylhs[4] = mxCreateDoubleScalar(4);
150
151     /* Call Matlab "plot" */
152     mexEvalString("figure(3)");
153     status = mexCallMATLAB(0, NULL, 5, xlhs, "plot");
154     if(status != 0)
155         mexErrMsgTxt("Plot failure.");
156     mexEvalString("figure(4)");
157     status = mexCallMATLAB(0, NULL, 5, ylhs, "plot");
158     if(status != 0)
159         mexErrMsgTxt("Plot failure.");
160
161     /* Destroy allocated memory */
162     mxDestroyArray(mxX1B);
163     mxDestroyArray(mxX3B);
164     mxDestroyArray(mxPARAMB);
165
166     printf("\n\nFinished plotting bifurcation diagrams.");
167     printf("\nBIFURMEX.C is TERMINATING ...\n");
168 }
169
```

B.5 Phase Movie

```
/*-------------------------------------------------------------------------
 * Phase plane movie function (phasemovmex.c)
 *  Loads time series data and truncates to last PHASESIZE data points.
 *  Determines maximum plot sizes for each phase plane and saves
 *  truncated data to phase file to avoid reloading entire sim data.
 *-------------------------------------------------------------------------
 *  Input Parameters:
 *      Name:       Type:       Connotation:
 *      -----       -----       ------------
 *      SIMFILES    mxArray*    Matlab cell array of sim datafile names
 *      PHASEFILES   mxArray*   Matlab cell array of phase datafile names
 *
 *  Output parameters:
 *      Name:       Type:       Connotation:
 *      -----       -----       ------------
 *      XPLOTSIZE   mxArray*    Matlab array of plotsizes for "axis"
 *      YPLOTSIZE   mxArray*    Matlab array of plotsizes for "axis"
 *
 *  Variables:
 *      Name:       Type:       Connotation:
 *      -----       -----       ------------
 *      size        int         number of file names in DATFILES
 *      xplotsize   double*     pointer to XPLOTSIZE
 *      yplotsize   double*     pointer to YPLOTSIZE
 *      mxFILE      mxArray*    Matlab filename string
 *      buflen      int         length of string buffer for file name
 *      simname     char*       C filename string
 *      status      int         function call status variable
 *      simmat      MATfile*    mat-file pointer
 *      mxX1        mxArray*    x1 values loaded from file
 *      mxX2        mxArray*    x2 values loaded from file
 *      mxX3        mxArray*    x3 values loaded from file
 *      mxX4        mxArray*    x4 values loaded from file
 *      x1          double*     pointer to mxX1
 *      x2          double*     pointer to mxX2
 *      x3          double*     pointer to mxX3
 *      x4          double*     pointer to mxX4
 *      iter        int         number data points loaded from files
 *      mxX1PP      mxArray*    truncated x1 values to write to phase file
 *      mxX2PP      mxArray*    truncated x2 values to write to phase file
 *      mxX3PP      mxArray*    truncated x3 values to write to phase file
 *      mxX4PP      mxArray*    truncated x4 values to write to phase file
 *      x1pp        double*     pointer to mxX1PP
 *      x2pp        double*     pointer to mxX2PP
 *      x3pp        double*     pointer to mxX3PP
 *      x4pp        double*     pointer to mxX4PP
```

```
47  *       phasename   char*       C filename string
48  *       phasemat    MATfile*    mat-file pointer
49  *       scale       double      scaling variable for plotsizes
50  *       i           int         counter
51  *       j           int         counter
52  *-------------------------------------------------------------------------*/
53  #include "mex.h"
54  #include "mat.h"
55  #include <stdio.h>
56  #include <stdlib.h>
57
58  #define SIMFILES    prhs[0]
59  #define PHASEFILES  prhs[1]
60  #define XPLOTSIZE   plhs[0]
61  #define YPLOTSIZE   plhs[1]
62  #define PHASESIZE   8001
63
64  void mexFunction( int nlhs, mxArray *plhs[],
65                    int nrhs, const mxArray *prhs[] )
66  {
67      int     size,i,j,buflen,status,iter;
68      double  *xplotsize,*yplotsize,*x1,*x2,*x3,*x4,*x1pp,*x2pp,*x3pp,*x4pp,
69              scale;
70      char    *simname,*phasename;
71      MATFile *simmat,*phasemat;
72      mxArray *mxFILE,*mxX1,*mxX2,*mxX3,*mxX4,*mxX1PP,*mxX2PP,*mxX3PP,*mxX4PP;
73
74      /* check for input. */
75      if(nrhs != 2)
76          mexErrMsgTxt("\n Invalid inputs \n\trequired: sim filename cell
77  ↪ array, phase filename cell array");
78
79      size = mxGetNumberOfElements(SIMFILES);
80      if(size != mxGetNumberOfElements(PHASEFILES))
81          mexErrMsgTxt("\n Improper number of phase files");
82
83      /* Create and reference output arrays */
84      XPLOTSIZE = mxCreateDoubleMatrix(4,1,mxREAL);
85      YPLOTSIZE = mxCreateDoubleMatrix(4,1,mxREAL);
86      xplotsize = mxGetPr(XPLOTSIZE);
87      yplotsize = mxGetPr(YPLOTSIZE);
88
89      /* Load each file and process */
90      for(i=0;i<size;i++) {
91          mxFILE = mxGetCell(SIMFILES,i);
92          buflen = (mxGetM(mxFILE) * mxGetN(mxFILE) * sizeof(mxChar)) + 1;
93          simname = mxCalloc(buflen, sizeof(char));
```

```
94          status = mxGetString(mxFILE, simname, buflen);
95          if(status != 0)
96              mexErrMsgTxt("Sim datafile name error.");
97
98          printf("\n Reading sim file:   %s ...", simname);
99
100         /* Open sim data file */
101         simmat = matOpen(simname, "r");
102         if(simmat == NULL) {
103             printf("\n Error opening file: %s", simname);
104             return; }
105
106         /* Read in each array and reference */
107         mxX1 = matGetVariable(simmat, "x1");
108         mxX2 = matGetVariable(simmat, "x2");
109         mxX3 = matGetVariable(simmat, "x3");
110         mxX4 = matGetVariable(simmat, "x4");
111         x1 = mxGetPr(mxX1);
112         x2 = mxGetPr(mxX2);
113         x3 = mxGetPr(mxX3);
114         x4 = mxGetPr(mxX4);
115         iter = mxGetM(mxX1);
116
117         /* Create and reference to new truncated arrays */
118         mxX1PP = mxCreateDoubleMatrix(PHASESIZE,1,mxREAL);
119         mxX2PP = mxCreateDoubleMatrix(PHASESIZE,1,mxREAL);
120         mxX3PP = mxCreateDoubleMatrix(PHASESIZE,1,mxREAL);
121         mxX4PP = mxCreateDoubleMatrix(PHASESIZE,1,mxREAL);
122         x1pp = mxGetPr(mxX1PP);
123         x2pp = mxGetPr(mxX2PP);
124         x3pp = mxGetPr(mxX3PP);
125         x4pp = mxGetPr(mxX4PP);
126
127         /* Truncate to last PHASESIZE data points */
128         for(j=0;j<PHASESIZE;j++) {
129             x1pp[j] = x1[iter-(PHASESIZE-j)];
130             x2pp[j] = x2[iter-(PHASESIZE-j)];
131             x3pp[j] = x3[iter-(PHASESIZE-j)];
132             x4pp[j] = x4[iter-(PHASESIZE-j)];
133         }
134
135         /* Destroy allocated memory */
136         mxDestroyArray(mxX1);
137         mxDestroyArray(mxX2);
138         mxDestroyArray(mxX3);
139         mxDestroyArray(mxX4);
140
```

```
141         /* Close sim data file */
142         if(matClose(simmat) != 0) {
143             printf("\n Error closing file: %s",simname);
144             return; }
145
146         if(i==0) {
147             /* Set initial plotsizes */
148             xplotsize[0]=x1pp[0];xplotsize[1]=x1pp[0];
149             xplotsize[2]=x2pp[0];xplotsize[3]=x2pp[0];
150             yplotsize[0]=x3pp[0];yplotsize[1]=x3pp[0];
151             yplotsize[2]=x4pp[0];yplotsize[3]=x4pp[0];
152             for(j=1;j<PHASESIZE;j++) {
153                 if(x1pp[j]<xplotsize[0]) {xplotsize[0]=x1pp[j];}
154                 if(x1pp[j]>xplotsize[1]) {xplotsize[1]=x1pp[j];}
155                 if(x2pp[j]<xplotsize[2]) {xplotsize[2]=x2pp[j];}
156                 if(x2pp[j]>xplotsize[3]) {xplotsize[3]=x2pp[j];}
157                 if(x3pp[j]<yplotsize[0]) {yplotsize[0]=x3pp[j];}
158                 if(x3pp[j]>yplotsize[1]) {yplotsize[1]=x3pp[j];}
159                 if(x4pp[j]<yplotsize[2]) {yplotsize[2]=x4pp[j];}
160                 if(x4pp[j]>yplotsize[3]) {yplotsize[3]=x4pp[j];}
161             }
162         }
163         else {
164             /* Get maximums and minimums for plotsizes */
165             for(j=0;j<PHASESIZE;j++) {
166                 if(x1pp[j]<xplotsize[0]) {xplotsize[0]=x1pp[j];}
167                 if(x1pp[j]>xplotsize[1]) {xplotsize[1]=x1pp[j];}
168                 if(x2pp[j]<xplotsize[2]) {xplotsize[2]=x2pp[j];}
169                 if(x2pp[j]>xplotsize[3]) {xplotsize[3]=x2pp[j];}
170                 if(x3pp[j]<yplotsize[0]) {yplotsize[0]=x3pp[j];}
171                 if(x3pp[j]>yplotsize[1]) {yplotsize[1]=x3pp[j];}
172                 if(x4pp[j]<yplotsize[2]) {yplotsize[2]=x4pp[j];}
173                 if(x4pp[j]>yplotsize[3]) {yplotsize[3]=x4pp[j];}
174             }
175         }
176
177         mxFILE = mxGetCell(PHASEFILES,i);
178         buflen = (mxGetM(mxFILE) * mxGetN(mxFILE) * sizeof(mxChar)) + 1;
179         phasename = mxCalloc(buflen, sizeof(char));
180         status = mxGetString(mxFILE, phasename, buflen);
181         if(status != 0)
182             mexErrMsgTxt("Phase datafile name error.");
183         printf("\n Writing phase file: %s ...", phasename);
184
185         /* Open phase data file */
186         phasemat = matOpen(phasename, "w");
187         if(phasemat == NULL) {
```

```
188             printf("\n Error opening file: %s", phasename);
189             return; }
190
191         /* Write phase data file */
192         status = matPutVariable(phasemat, "x1", mxX1PP);
193         if(status != 0) {
194             printf("\n Error writing x1 to %s",phasename);
195             return; }
196         else { mxDestroyArray(mxX1PP); }
197         status = matPutVariable(phasemat, "x2", mxX2PP);
198         if(status != 0) {
199             printf("\n Error writing x2 to %s",phasename);
200             return; }
201         else { mxDestroyArray(mxX2PP); }
202         status = matPutVariable(phasemat, "x3", mxX3PP);
203         if(status != 0) {
204             printf("\n Error writing x3 to %s",phasename);
205             return; }
206         else { mxDestroyArray(mxX3PP); }
207         status = matPutVariable(phasemat, "x4", mxX4PP);
208         if(status != 0) {
209             printf("\n Error writing x4 to %s",phasename);
210             return; }
211         else { mxDestroyArray(mxX4PP); }
212
213         /* Close phase file */
214         if(matClose(phasemat) != 0) {
215             printf("\n Error closing file: %s",phasename);
216             return; }
217     }
218
219     /* Rescale to add 10% border */
220     scale = xplotsize[1]-xplotsize[0];
221     xplotsize[1] += scale*0.1;
222     xplotsize[0] -= scale*0.1;
223     scale = xplotsize[3]-xplotsize[2];
224     xplotsize[3] += scale*0.1;
225     xplotsize[2] -= scale*0.1;
226     scale = yplotsize[1]-yplotsize[0];
227     yplotsize[1] += scale*0.1;
228     yplotsize[0] -= scale*0.1;
229     scale = yplotsize[3]-yplotsize[2];
230     yplotsize[3] += scale*0.1;
231     yplotsize[2] -= scale*0.1;
232
233     printf("\n\n Finished calculating maximum figure dimensions and writing
234 ↪ phase data files.");
```

```
235         printf("\n PHASEMOVMEX.C is TERMINATING ...\n");
236     }
```

B.6 Poincaré Map Movie

```
1  /*-------------------------------------------------------------------------
2   * Poincare map movie function (poinmovmex.c)
3   *  Loads poincare map data and truncated to remove transient data points.
4   *  Determines maximum plot sizes for each poincare map and saves
5   *  truncated data to temp file to avoid reloading entire poincare data.
6   *-------------------------------------------------------------------------
7   *  Input Parameters:
8   *       Name:        Type:        Connotation:
9   *       -----        -----        ------------
10  *       CIGNORE      mxArray*     Matlab number of cycles to ignore
11  *       DATFILES     mxArray*     Matlab cell array of sim datafile names
12  *       TEMPFILES    mxArray*     Matlab cell array of temp datafile names
13  *
14  *  Output parameters:
15  *       Name:        Type:        Connotation:
16  *       -----        -----        ------------
17  *       XPLOTSIZE    mxArray*     Matlab array of plotsizes for "axis"
18  *       YPLOTSIZE    mxArray*     Matlab array of plotsizes for "axis"
19  *       PERIODS      mxArray*     number of periods of data in files
20  *
21  *  Variables:
22  *       Name:        Type:        Connotation:
23  *       -----        -----        ------------
24  *       c_ignore     double       number of cycles to ignore
25  *       igcycles     int          number of cycles to ignore (as integer)
26  *       size         int          number of file names in DATFILES
27  *       xplotsize    double*      pointer to XPLOTSIZE
28  *       yplotsize    double*      pointer to YPLOTSIZE
29  *       mxFILE       mxArray*     Matlab filename string
30  *       buflen       int          length of string buffer for file name
31  *       filename     char*        C filename string
32  *       status       int          function call status variable
33  *       filemat      MATfile*     mat-file pointer
34  *       mxX1P        mxArray*     x1p values loaded from file
35  *       mxX2P        mxArray*     x2p values loaded from file
36  *       mxX3P        mxArray*     x3p values loaded from file
37  *       mxX4P        mxArray*     x4p values loaded from file
38  *       x1p          double*      pointer to mxX1P
39  *       x2p          double*      pointer to mxX2P
40  *       x3p          double*      pointer to mxX3P
41  *       x4p          double*      pointer to mxX4P
42  *       cycles       int          number of cycles loaded from files
43  *       mxX1PT       mxArray*     truncated x1p values to write to temp file
44  *       mxX2PT       mxArray*     truncated x2p values to write to temp file
45  *       mxX3PT       mxArray*     truncated x3p values to write to temp file
46  *       mxX4PT       mxArray*     truncated x4p values to write to temp file
```

```
47  *       x1pt        double*     pointer to mxX1PT
48  *       x2pt        double*     pointer to mxX2PT
49  *       x3pt        double*     pointer to mxX3PT
50  *       x4pt        double*     pointer to mxX4PT
51  *       tempname    char*       C filename string
52  *       tempmat     MATfile*    mat-file pointer
53  *       scale       double      scaling variable for plotsizes
54  *       i           int         counter
55  *       j           int         counter
56  *-----------------------------------------------------------------------*/
57
58  #include "mex.h"
59  #include "mat.h"
60  #include <stdio.h>
61  #include <stdlib.h>
62  #include <math.h>
63
64  #define CIGNORE     prhs[0]
65  #define DATFILES    prhs[1]
66  #define TEMPFILES   prhs[2]
67  #define XPLOTSIZE   plhs[0]
68  #define YPLOTSIZE   plhs[1]
69  #define PERIODS     plhs[2]
70
71  void mexFunction( int nlhs, mxArray *plhs[],
72                    int nrhs, const mxArray *prhs[] )
73  {
74      int     igcycles,size,i,j,buflen,status,cycles;
75      double  c_ignore,*xplotsize,*yplotsize,*x1p,*x2p,*x3p,*x4p,*x1pt,*x2pt,
76              *x3pt,*x4pt,scale;
77      char    *filename,*tempname;
78      MATFile *filemat,*tempmat;
79      mxArray *mxFILE,*mxX1P,*mxX2P,*mxX3P,*mxX4P,*mxX1PT,*mxX2PT,*mxX3PT,
80              *mxX4PT;
81
82      /* check for input. */
83      if(nrhs != 3)
84          mexErrMsgTxt("\n Invalid inputs \n\trequired: number of cycles to
85  ↪ ignore, filename cell array, tempfilename cell array");
86
87      /* Get inputs */
88      c_ignore = mxGetScalar(CIGNORE);
89      /* Convert to integer for use in array indexing */
90      igcycles = ceil(c_ignore);
91
92      size = mxGetNumberOfElements(DATFILES);
93      if(size != mxGetNumberOfElements(TEMPFILES))
```

```
94          mexErrMsgTxt("\n Improper number of temporary files");
95
96      /* Create and reference output arrays */
97      XPLOTSIZE = mxCreateDoubleMatrix(4,1,mxREAL);
98      YPLOTSIZE = mxCreateDoubleMatrix(4,1,mxREAL);
99      xplotsize = mxGetPr(XPLOTSIZE);
100     yplotsize = mxGetPr(YPLOTSIZE);
101
102     /* Load each file and process */
103     for(i=0;i<size;i++) {
104         mxFILE = mxGetCell(DATFILES,i);
105         buflen = (mxGetM(mxFILE) * mxGetN(mxFILE) * sizeof(mxChar)) + 1;
106         filename = mxCalloc(buflen, sizeof(char));
107         status = mxGetString(mxFILE, filename, buflen);
108         if(status != 0)
109             mexErrMsgTxt("Poincare datafile name error.");
110
111         printf("\n Reading poincare file: %s ...", filename);
112
113         /* Open poincare data file */
114         filemat = matOpen(filename, "r");
115         if(filemat == NULL) {
116             printf("\n Error opening file: %s", filename);
117             return; }
118
119         /* Read in each array and reference */
120         mxX1P = matGetVariable(filemat, "x1p");
121         mxX2P = matGetVariable(filemat, "x2p");
122         mxX3P = matGetVariable(filemat, "x3p");
123         mxX4P = matGetVariable(filemat, "x4p");
124         x1p = mxGetPr(mxX1P);
125         x2p = mxGetPr(mxX2P);
126         x3p = mxGetPr(mxX3P);
127         x4p = mxGetPr(mxX4P);
128         cycles = mxGetM(mxX1P);
129
130         /* Error check for ignored cycles */
131         if(cycles<c_ignore)
132             mexErrMsgTxt("Ignored cycles are greater than available data!");
133
134         /* Create and reference to new truncated arrays */
135         mxX1PT = mxCreateDoubleMatrix((cycles-igcycles),1,mxREAL);
136         mxX2PT = mxCreateDoubleMatrix((cycles-igcycles),1,mxREAL);
137         mxX3PT = mxCreateDoubleMatrix((cycles-igcycles),1,mxREAL);
138         mxX4PT = mxCreateDoubleMatrix((cycles-igcycles),1,mxREAL);
139         x1pt = mxGetPr(mxX1PT);
140         x2pt = mxGetPr(mxX2PT);
```

```
141         x3pt = mxGetPr(mxX3PT);
142         x4pt = mxGetPr(mxX4PT);
143
144         /* Truncate to remove transient cycles to fill temp data */
145         for(j=0;j<(cycles-igcycles);j++) {
146             x1pt[j] = x1p[j+igcycles];
147             x2pt[j] = x2p[j+igcycles];
148             x3pt[j] = x3p[j+igcycles];
149             x4pt[j] = x4p[j+igcycles];
150         }
151
152         /* Destroy allocated memory */
153         mxDestroyArray(mxX1P);
154         mxDestroyArray(mxX2P);
155         mxDestroyArray(mxX3P);
156         mxDestroyArray(mxX4P);
157
158         /* Close poincare data file */
159         if(matClose(filemat) != 0) {
160             printf("\n Error closing file: %s",filename);
161             return; }
162
163         if(i==0) {
164             /* Set initial plotsizes */
165             xplotsize[0]=x1pt[0];xplotsize[1]=x1pt[0];
166             xplotsize[2]=x2pt[0];xplotsize[3]=x2pt[0];
167             yplotsize[0]=x3pt[0];yplotsize[1]=x3pt[0];
168             yplotsize[2]=x4pt[0];yplotsize[3]=x4pt[0];
169             for(j=1;j<(cycles-igcycles);j++) {
170                 if(x1pt[j]<xplotsize[0]) {xplotsize[0]=x1pt[j];}
171                 if(x1pt[j]>xplotsize[1]) {xplotsize[1]=x1pt[j];}
172                 if(x2pt[j]<xplotsize[2]) {xplotsize[2]=x2pt[j];}
173                 if(x2pt[j]>xplotsize[3]) {xplotsize[3]=x2pt[j];}
174                 if(x3pt[j]<yplotsize[0]) {yplotsize[0]=x3pt[j];}
175                 if(x3pt[j]>yplotsize[1]) {yplotsize[1]=x3pt[j];}
176                 if(x4pt[j]<yplotsize[2]) {yplotsize[2]=x4pt[j];}
177                 if(x4pt[j]>yplotsize[3]) {yplotsize[3]=x4pt[j];}
178             }
179         }
180         else {
181             /* Get maximums and minimums for plotsizes */
182             for(j=0;j<(cycles-igcycles);j++) {
183                 if(x1pt[j]<xplotsize[0]) {xplotsize[0]=x1pt[j];}
184                 if(x1pt[j]>xplotsize[1]) {xplotsize[1]=x1pt[j];}
185                 if(x2pt[j]<xplotsize[2]) {xplotsize[2]=x2pt[j];}
186                 if(x2pt[j]>xplotsize[3]) {xplotsize[3]=x2pt[j];}
187                 if(x3pt[j]<yplotsize[0]) {yplotsize[0]=x3pt[j];}
```

```
188                 if(x3pt[j]>yplotsize[1]) {yplotsize[1]=x3pt[j];}
189                 if(x4pt[j]<yplotsize[2]) {yplotsize[2]=x4pt[j];}
190                 if(x4pt[j]>yplotsize[3]) {yplotsize[3]=x4pt[j];}
191             }
192         }
193
194         mxFILE = mxGetCell(TEMPFILES,i);
195         buflen = (mxGetM(mxFILE) * mxGetN(mxFILE) * sizeof(mxChar)) + 1;
196         tempname = mxCalloc(buflen, sizeof(char));
197         status = mxGetString(mxFILE, tempname, buflen);
198         if(status != 0)
199             mexErrMsgTxt("Temp datafile name error.");
200         printf("\n Writing tempfile:      %s ...", tempname);
201
202         /* Open temp data file */
203         tempmat = matOpen(tempname, "w");
204         if(tempmat == NULL) {
205             printf("\n Error opening file: %s", tempname);
206             return; }
207
208         /* Write temp data file */
209         status = matPutVariable(tempmat, "x1p", mxX1PT);
210         if(status != 0) {
211             printf("\n Error writing x1p to %s",tempname);
212             return; }
213         else { mxDestroyArray(mxX1PT); }
214         status = matPutVariable(tempmat, "x2p", mxX2PT);
215         if(status != 0) {
216             printf("\n Error writing x2p to %s",tempname);
217             return; }
218         else { mxDestroyArray(mxX2PT); }
219         status = matPutVariable(tempmat, "x3p", mxX3PT);
220         if(status != 0) {
221             printf("\n Error writing x3p to %s",tempname);
222             return; }
223         else { mxDestroyArray(mxX3PT); }
224         status = matPutVariable(tempmat, "x4p", mxX4PT);
225         if(status != 0) {
226             printf("\n Error writing x4p to %s",tempname);
227             return; }
228         else { mxDestroyArray(mxX4PT); }
229
230         /* Close temp file */
231         if(matClose(tempmat) != 0) {
232             printf("\n Error closing file: %s",tempname);
233             return; }
234     }
```

```
235     /* Get output */
236     PERIODS = mxCreateDoubleScalar(cycles-1);
237
238     /* Rescale to add 10% border */
239     scale = xplotsize[1]-xplotsize[0];
240     xplotsize[1] += scale*0.1;
241     xplotsize[0] -= scale*0.1;
242     scale = xplotsize[3]-xplotsize[2];
243     xplotsize[3] += scale*0.1;
244     xplotsize[2] -= scale*0.1;
245     scale = yplotsize[1]-yplotsize[0];
246     yplotsize[1] += scale*0.1;
247     yplotsize[0] -= scale*0.1;
248     scale = yplotsize[3]-yplotsize[2];
249     yplotsize[3] += scale*0.1;
250     yplotsize[2] -= scale*0.1;
251
252     printf("\n\n Finished calculating maximum figure dimensions and writing
253 ↪ temp files.");
254     printf("\n POINMOVMEX.C is TERMINATING ...\n");
255 }
```

Appendix C

Matlab Programs

C.1 Solver Comparisons for Coupled Mass System

```
%-------------------------------------------------------------------------
% Comparison of solutions methods to analytic solution of differential
% equations for coupled mass system (coupmass_comp.m)
%
% By Joseph O'Day 2005
%-------------------------------------------------------------------------
%        ..        .
%   [m]{x} + [c]{x} + [k]{x} = {f}
%
%   [m] = [m 0; 0 m]
%   [c] = [3c -c;-c 3c]
%   [k] = [2k -k;-k 2k]
%   {f} = {f*sin(w*t);0}
%-------------------------------------------------------------------------
function main()
clear
clc
close all
global tSpan te x1e x2e ci

fprintf(1,'\n-------------------------------------------------------------
↪ ------------');
fprintf(1,'\n Solving Method Comparison by Joseph O''Day');
fprintf(1,'\n-------------------------------------------------------------
↪ ------------');
today = date;
now = clock;
fprintf(1,'\n Run Date: %s',today);
fprintf(1,'\n Run Time: %i:%i',now(4),now(5));

% Set system parameters
```

```
32 m = 2;
33 k = 20;
34 c = 0.2*sqrt(k*m); % ensure underdamped system
35 f = 100;
36 w = 15;
37 % Set initial conditions
38 IC = [0 0 0 0];
39 % load model parameters
40 simparam(m,c,k,f,w,IC);
41
42 % Set solve options
43 %   time span
44 te=0:0.01:20;
45 N = length(te);
46 tSpan = [0 te(N)];
47 %   error tolerances for variable step solvers
48 relTol = 1e-3;
49 absTol = 1e-6;
50 refine = 1;
51 varoptions = odeset('Reltol',relTol,'AbsTol',absTol,'Refine',refine);
52 %   step size for fixed solve methods
53 fixedStep = .01;
54
55 % Print system parameters, ICs, time span
56 fprintf(1,'\n\n System Parameters:');
57 fprintf(1,'\n\t m = %.3f',m);
58 fprintf(1,'\n\t c = %.3f',c);
59 fprintf(1,'\n\t k = %.3f',k);
60 fprintf(1,'\n\t f = %.3f',f);
61 fprintf(1,'\n\t w = %.3f',w);
62 fprintf(1,'\n\n Initial Conditions:');
63 fprintf(1,'\n\t x1    = %.3f',IC(1));
64 fprintf(1,'\n\t x1dot = %.3f',IC(2));
65 fprintf(1,'\n\t x2    = %.3f',IC(3));
66 fprintf(1,'\n\t x2dot = %.3f',IC(4));
67 fprintf(1,'\n\n Time Span: 0-%d seconds',te(N));
68
69 % Set color index
70 ci = 0;
71
72 %-------------------------------------------------------------------------
73 % Analytic Solution (underdamped)
74 %-------------------------------------------------------------------------
75 wn1 = sqrt(k/m);
76 wn2 = sqrt(3*k/m);
77 zeta1 = c/sqrt(k*m);
78 zeta2 = 2*c/sqrt(3*k*m);
```

```
wd1 = wn1*sqrt(1-zeta1^2);
wd2 = wn2*sqrt(1-zeta2^2);

phi = [1 1;1 -1];

dum = inv(phi)*[IC(1) IC(3)]';
qIC(1) = dum(1);
qIC(3) = dum(2);
dum = inv(phi)*[IC(2) IC(4)]';
qIC(2) = dum(1);
qIC(4) = dum(2);
F = [f/(2*m) 0]';

A = [wn1^2-w^2 2*zeta1*wn1*w;-2*zeta1*wn1*w wn1^2-w^2];
dum = inv(A)*F;
C1 = dum(1);
D1 = dum(2);
A1 = qIC(1)-C1;
B1 = (qIC(2)+A1*zeta1*wn1-D1*w)/wd1;
q1 = A1.*exp(-zeta1.*wn1.*te).*cos(wd1.*te)+B1.*exp(-zeta1.*wn1.*te).*
↪ sin(wd1.*te)+C1.*cos(w.*te)+D1.*sin(w.*te);

A = [wn2^2-w^2 2*zeta2*wn2*w;-2*zeta2*wn2*w wn2^2-w^2];
dum= inv(A)*F;
C2 = dum(1);
D2 = dum(2);
A2 = qIC(3)-C2;
B2 = (qIC(4)+A2*zeta2*wn2-D2*w)/wd2;
q2 = A2.*exp(-zeta2.*wn2.*te).*cos(wd2.*te)+B2.*exp(-zeta2.*wn2.*te).*
↪ sin(wd2.*te)+C2.*cos(w.*te)+D2.*sin(w.*te);

x1e = q1+q2;
x2e = q1-q2;

%plot solution
figure(1);
hold on;
plot(te,x1e,'b-');
figure(2);
hold on;
plot(te,x2e,'b-');

%-------------------------------------------------------------------------
% Compute Approximate Solution using Matlab ODE solvers
%-------------------------------------------------------------------------
% ODE45
tic;
```

```
126 for i=1:20;
127     [t45,x45] = ode45(@coupmasseq,tSpan,IC,varoptions,m,c,k,f,w);
128 end
129 time = toc;
130 parameters.solver = 'ODE45';
131 parameters.solvertype = 'variable_step';
132 parameters.reltol = varoptions.RelTol;
133 parameters.abstol = varoptions.AbsTol;
134 parameters.refine = varoptions.Refine;
135 outputs(parameters,time,t45,x45);
136 clear t45 x45;
137
138 % ODE23
139 tic;
140 for i=1:20;
141     [t23,x23] = ode23(@coupmasseq,tSpan,IC,varoptions,m,c,k,f,w);
142 end
143 time = toc;
144 parameters.solver = 'ODE23';
145 parameters.solvertype = 'variable_step';
146 parameters.reltol = varoptions.RelTol;
147 parameters.abstol = varoptions.AbsTol;
148 parameters.refine = varoptions.Refine;
149 outputs(parameters,time,t23,x23);
150 clear t23 x23;
151
152 % ODE113
153 tic;
154 for i=1:20;
155     [t113,x113] = ode113(@coupmasseq,tSpan,IC,varoptions,m,c,k,f,w);
156 end
157 time = toc;
158 parameters.solver = 'ODE113';
159 parameters.solvertype = 'variable_step';
160 parameters.reltol = varoptions.RelTol;
161 parameters.abstol = varoptions.AbsTol;
162 parameters.refine = varoptions.Refine;
163 outputs(parameters,time,t113,x113);
164 clear t113 x113;
165
166 % ODE5
167 fixoptions = simset('Solver','ode5','FixedStep',fixedStep);
168 tic;
169 for i=1:20;
170     sim('coupmassmod',tSpan,fixoptions);
171 end
172 time = toc;
```

```
parameters.solver = 'ODE5';
parameters.solvertype = 'fixed_step';
parameters.size = fixoptions.FixedStep;
X = [x1 x2 x3 x4];
outputs(parameters,time,t,X);
clear t x;

% ODE4
fixoptions = simset('Solver','ode4','FixedStep',fixedStep);
tic;
for i=1:20;
    sim('coupmassmod',tSpan,fixoptions);
end
time = toc;
parameters.solver = 'ODE4';
parameters.solvertype = 'fixed_step';
parameters.size = fixoptions.FixedStep;
X = [x1 x2 x3 x4];
outputs(parameters,time,t,X);
clear t x;

% ODE3
fixoptions = simset('Solver','ode3','FixedStep',fixedStep);
tic;
for i=1:20;
    sim('coupmassmod',tSpan,fixoptions);
end
time = toc;
parameters.solver = 'ODE3';
parameters.solvertype = 'fixed_step';
parameters.size = fixoptions.FixedStep;
X = [x1 x2 x3 x4];
outputs(parameters,time,t,X);
clear t x;

close_system('coupmassmod');

%-------------------------------------------------------------------------
% Compute solution using 4th order Runge-Kutta m-file
%-------------------------------------------------------------------------
tic;
for i=1:20;
    X=zeros(te(N)/fixedStep+1,4);
    t=zeros(te(N)/fixedStep+1,1);
    X0 = [IC(1) IC(2) IC(3) IC(4)];
    h = fixedStep;
    for i=1:te(N)/fixedStep,
```

```
220         x = X0;
221         k1 = h*coupmasseq(t(i),x,m,c,k,f,w);
222         x = X0+1/2*k1';
223         k2 = h*coupmasseq(t(i)+h/2,x,m,c,k,f,w);
224         x = X0+1/2*k2';
225         k3 = h*coupmasseq(t(i)+h/2,x,m,c,k,f,w);
226         x = X0+k3';
227         k4 = h*coupmasseq(t(i)+h,x,m,c,k,f,w);
228         t(i+1) = t(i)+h;
229         X(i+1,:) = X(i,:)+1/6.*(k1'+2*k2'+2*k3'+k4');
230         X0 = X(i+1,:);
231     end
232 end
233 time = toc;
234 parameters.solver = '4th order Runge-Kutta m-file';
235 parameters.solvertype = 'fixed_step';
236 parameters.size = fixedStep;
237 outputs(parameters,time,t,X);
238 clear X0 X t x k1 k2 k3 k4;
239
240 %-------------------------------------------------------------------------
241 % Compute solution using 4th order Runge-Kutta mex-file
242 %-------------------------------------------------------------------------
243 tic;
244 for i=1:20;
245     X0 = [IC(1) IC(2) IC(3) IC(4)];
246     h = fixedStep;
247     [t,x1,x2,x3,x4] = coupmassRK4mex(te(N),fixedStep,X0,m,c,k,f,w);
248 end
249 time=toc;
250 parameters.solver = '4th order Runge-Kutta mex-file';
251 parameters.solvertype = 'fixed_step';
252 parameters.size = fixedStep;
253 X=[x1,x2,x3,x4];
254 outputs(parameters,time,t,X);
255 clear X0 X t;
256
257 %-------------------------------------------------------------------------
258 % Compute solution using Lie Series approximations m-file
259 %-------------------------------------------------------------------------
260 % For orders 2 through 6
261 for o=2:6,
262     tic;
263     for i=1:20;
264         X=zeros(te(N)/fixedStep+1,5);
265         X0 = [IC(1) IC(2) IC(3) IC(4) te(1)];
266         for i=1:te(N)/fixedStep,
```

```
            NX = coupmasslie(o,X0,fixedStep,m,c,k,f,w);
            X(i+1,:) = NX;
            X0 = NX;
        end
    end
    time=toc;
    parameters.solver = 'Lie Series m-file';
    parameters.solvertype = 'lie';
    parameters.order = o;
    parameters.size = fixedStep;
    outputs(parameters,time,X(:,5),X(:,1:4));
    clear X0 X NX;
end

%-------------------------------------------------------------------------
% Compute solution using Lie Series approximations mex-file
%-------------------------------------------------------------------------
for o=2:6,
    tic;
    for i=1:20;
        X0 = [IC(1) IC(2) IC(3) IC(4) te(1)];
        [x1 x2 x3 x4 x5] = coupmassliemex(o,te(N),fixedStep,X0,m,c,k,f,w);
    end
    time=toc;
    parameters.solver = 'Lie Series mex-file';
    parameters.solvertype = 'lie';
    parameters.order = o;
    parameters.size = fixedStep;
    outputs(parameters,time,x5,[x1 x2 x3 x4]);
    clear x1 x2 x3 x4 x5;
end

fprintf(1,'\n\n\n Normal Termination of COUPMASS_COMP.M');
fprintf(1,'\n-------------------------------------------------------------
↪ ------------');

return;

%-------------------------------------------------------------------------
% Set Simulink model parameters
%-------------------------------------------------------------------------
function simparam(m,c,k,f,w,IC)

open_system('coupmassmod');
set_param('coupmassmod/Gain1','Gain',num2str(1/m));
set_param('coupmassmod/Gain2','Gain',num2str(1/m));
set_param('coupmassmod/Gain3','Gain',num2str(c));
```

```
314 set_param('coupmassmod/Gain4','Gain',num2str(c));
315 set_param('coupmassmod/Gain5','Gain',num2str(k));
316 set_param('coupmassmod/Gain6','Gain',num2str(k));
317 set_param('coupmassmod/Constant1','Value',num2str(f));
318 set_param('coupmassmod/Constant2','Value',num2str(w));
319 set_param('coupmassmod/Integrator','InitialCondition',num2str(IC(2)));
320 set_param('coupmassmod/Integrator1','InitialCondition',num2str(IC(1)));
321 set_param('coupmassmod/Integrator2','InitialCondition',num2str(IC(4)));
322 set_param('coupmassmod/Integrator3','InitialCondition',num2str(IC(3)));
323 save_system('coupmassmod');
324 close_system('coupmassmod');
325
326 return;
327 %-------------------------------------------------------------------------
328 % State equations for ODE solvers
329 %-------------------------------------------------------------------------
330 function dxdt = coupmasseq(t,x,m,c,k,f,w)
331
332 dxdt = zeros(4,1);
333 dxdt(1) = x(2);
334 dxdt(2) = 1/m*(f*cos(w*t)-3*c*x(2)+c*x(4)-2*k*x(1)+k*x(3));
335 dxdt(3) = x(4);
336 dxdt(4) = 1/m*(-3*c*x(4)+c*x(2)-2*k*x(3)+k*x(1));
337
338 return;
339
340 %-------------------------------------------------------------------------
341 % Lie Series approximating equations
342 %-------------------------------------------------------------------------
343 function NX = coupmasslie(order,X,step,m,c,k,f,w)
344
345 z1 = X(1);
346 z2 = X(2);
347 z3 = X(3);
348 z4 = X(4);
349 z5 = X(5);
350
351 h(1) = step;
352 h(2) = step^2/2;
353 h(3) = step^3/6;
354 h(4) = step^4/24;
355 h(5) = step^5/120;
356 h(6) = step^6/720;
357
358 U = zeros(6,5);
359
360 t1 = 1/m;
```

```
361  t2 = w*z5;
362  t3 = cos(t2);
363  t4 = f*t3;
364  t5 = c*z2;
365  t7 = c*z4;
366  t8 = k*z1;
367  t10 = k*z3;
368  t11 = t4-3.0*t5+t7-2.0*t8+t10;
369  t12 = t1*t11;
370  t15 = -3.0*t7+t5-2.0*t10+t8;
371  t16 = t1*t15;
372  t17 = t1*k;
373  t18 = t17*z2;
374  t20 = m*m;
375  t21 = 1/t20;
376  t22 = t21*c;
377  t23 = t22*t11;
378  t25 = t17*z4;
379  t26 = t22*t15;
380  t27 = t1*f;
381  t28 = sin(t2);
382  t31 = -2.0*t18-3.0*t23+t25+t26-t27*t28*w;
383  t34 = t18+t23-2.0*t25-3.0*t26;
384  t36 = t22*k*z2;
385  t39 = c*c;
386  t40 = t21*t39;
387  t43 = (-2.0*t17+10.0*t40)*t1;
388  t46 = t22*k*z4;
389  t50 = (-6.0*t40+t17)*t1;
390  t52 = f*t28;
391  t53 = t52*w;
392  t54 = t22*t53;
393  t56 = w*w;
394  t59 = 7.0*t36+t43*t11-5.0*t46+t50*t15+3.0*t54-t27*t3*t56;
395  t64 = -5.0*t36+t50*t11+7.0*t46+t43*t15-t54;
396  t65 = t43*k;
397  t66 = 2.0*t65;
398  t67 = t50*k;
399  t68 = -t66+t67;
400  t70 = t22*k;
401  t72 = t43*c;
402  t74 = t50*c;
403  t76 = (7.0*t70-3.0*t72+t74)*t1;
404  t78 = 2.0*t67;
405  t79 = t65-t78;
406  t84 = (t72-5.0*t70-3.0*t74)*t1;
407  t87 = t4*t56;
```

```
t88 = t22*t87;
t90 = t56*w;
t93 = t68*z2+t76*t11+t79*z4+t84*t15-t43*t53+3.0*t88+t27*t28*t90;
t99 = t79*z2+t84*t11+t68*z4+t76*t15-t50*t53-t88;
t100 = t76*k;
t101 = 2.0*t100;
t102 = t84*k;
t103 = -t101+t102;
t105 = t76*c;
t107 = t84*c;
t109 = (-t66+t67-3.0*t105+t107)*t1;
t111 = 2.0*t102;
t112 = t100-t111;
t116 = (t105+t65-t78-3.0*t107)*t1;
t120 = t52*t90;
t121 = t22*t120;
t123 = t56*t56;
t126 = t103*z2+t109*t11+t112*z4+t116*t15-t76*t53-t43*t87-3.0*t121+t27*t3*t123;
t133 = t112*z2+t116*t11+t103*z4+t109*t15-t84*t53-t50*t87+t121;
t134 = t109*k;
t136 = t116*k;
t137 = -2.0*t134+t136;
t139 = t109*c;
t141 = t116*c;
t143 = (-t101+t102-3.0*t139+t141)*t1;
t146 = t134-2.0*t136;
t150 = (t139+t100-t111-3.0*t141)*t1;
t156 = t22*t4*t123;

if order>6,
    error('\n Order of approximation too large!');
    return;
else
    if order>=1,
        U(1,1) = z2;
        U(1,2) = t12;
        U(1,3) = z4;
        U(1,4) = t16;
        U(1,5) = 1.0;
    end
    if order>=2,
        U(2,1) = t12;
        U(2,2) = t31;
        U(2,3) = t16;
        U(2,4) = t34;
        U(2,5) = 0.0;
    end
```

```
455       if order>=3,
456           U(3,1) = t31;
457           U(3,2) = t59;
458           U(3,3) = t34;
459           U(3,4) = t64;
460           U(3,5) = 0.0;
461       end
462       if order>=4,
463           U(4,1) = t59;
464           U(4,2) = t93;
465           U(4,3) = t64;
466           U(4,4) = t99;
467           U(4,5) = 0.0;
468       end
469       if order>=5,
470           U(5,1) = t93;
471           U(5,2) = t126;
472           U(5,3) = t99;
473           U(5,4) = t133;
474           U(5,5) = 0.0;
475       end
476       if order==6,
477           U(6,1) = t126;
478           U(6,2) = t137*z2+t143*t11+t146*z4+t150*t15-t109*t53-t76*t87+t43*
479   ↪ t120-3.0*t156-t27*t28*t123*w;
480           U(6,3) = t133;
481           U(6,4) = t146*z2+t150*t11+t137*z4+t143*t15-t116*t53-t84*t87+t50*
482   ↪  t120+t156;
483           U(6,5) = 0.0;
484       end
485
486       NX(1) = z1;
487       NX(2) = z2;
488       NX(3) = z3;
489       NX(4) = z4;
490       NX(5) = z5;
491
492       for i=1:5,
493           for j=1:order,
494               NX(i) = NX(i)+U(j,i)*h(j);
495           end
496       end
497   end
498
499   return;
500
501   %-----------------------------------------------------------------------
```

```
% Solution outputs
%-------------------------------------------------------------------------
function outputs(parameters,time,t,x)
global tSpan te x1e x2e ci

% color rotation
ci = ci+1;
dum = mod(ci,6);
if dum == 0,
    ci = 6;
elseif ci > 6,
    ci = dum;
end
colors = ['g' 'r' 'c' 'm' 'y' 'k'];

% Print outputs
fprintf(1,'\n\n\n %s solution', parameters.solver);
fprintf(1,'\n ------------------------------------');
fprintf(1,'\n Time required: %d minutes %.4f seconds',floor(time/20/60),
↪ time/20-floor(time/20/60)*60);

switch parameters.solvertype
    case 'variable_step'
        fprintf(1,'\n Relative Tolerance: %.1e', parameters.reltol);
        fprintf(1,'\n Absolute Tolerance: %.1e', parameters.abstol);
        fprintf(1,'\n Refine factor: %d', parameters.refine);
        step = te(2)-te(1);
        for i=1:length(t),
            index = round(t(i)/step)+1;
            terr(i) = te(index);
            x1err(i) = x1e(index);
            x2err(i) = x2e(index);
        end
    case 'fixed_step'
        fprintf(1,'\n Step size: %.2e', parameters.size);
        terr = te;
        x1err = x1e;
        x2err = x2e;
    case 'lie'
        fprintf(1,'\n Order: %d', parameters.order);
        fprintf(1,'\n Step size: %.2e', parameters.size);
        terr = te;
        x1err = x1e;
        x2err = x2e;
    otherwise
        fprintf(1,'\n Error!')
end
```

```
549
550 n = length(terr);
551 for i=1:n,
552     x1error(i)=abs((x1err(i)-x(i,1)));
553     x2error(i)=abs((x2err(i)-x(i,3)));
554 end
555
556 x1maxerr = max(x1error);
557 x1toterr = sqrt(sum(x1error.^2));
558 fprintf(1,'\n Mass 1 Max Error: %.3e',x1maxerr);
559 fprintf(1,'\n Mass 1 Total Error: %.3e',x1toterr);
560
561 x2maxerr = max(x2error);
562 x2toterr = sqrt(sum(x2error.^2));
563 fprintf(1,'\n Mass 2 Max Error: %.3e',x2maxerr);
564 fprintf(1,'\n Mass 2 Total Error: %.3e',x2toterr);
565
566 % plot solution and error
567 figure(1);
568 hold on;
569 plot(t,x(:,1),colors(ci));
570 figure(2);
571 hold on;
572 plot(t,x(:,3),colors(ci));
573 figure(3);
574 hold on
575 plot(terr,x1error,colors(ci));
576 figure(4);
577 hold on
578 plot(terr,x2error,colors(ci));
579
580 return;
```

C.2 Main Program for Synchronous Forcing

```
%-------------------------------------------------------------------------
% Main function routine for simulation of nonlinearly coupled Duffing
%   Oscillators with synchronous forcing (sync.m)
% By Joseph O'Day 2005
%-------------------------------------------------------------------------
% flag options:
%   'sim'       simulates system and also calculates Lyapunov exponents
%                   from Lie series Jacobian
%   'poincare'  calculates Poincare mapping for stroboscopic sampling
%                   based on input data
%   'freq'      calculates frequency spectrum based on input data via
%                   Welch's average modified periodogram method (psd.m)
%   'lyapunov'  creates Lyapunov spectrum from generated data
%   'bifur'     creates bifurcation diagram based on poincare map data
%   'phasemov'  creates a movie based on steady state phase plane
%   'freqmov'   creates a movie based on individual frequency spectrums
%   'poinmov'   creates a movie of Poincare diagrams (2D bifurcation
%                  diagram) and a movie of eigenvalue migration
%
% variable argument requirements:
%   'sim'       (time span,step size,IC vector[x1(0) x2(0) x3(0) x4(0)],
%                   ignored time before Lyap. spectrum,z,a,b,c,d,e,f)
%   'poincare'  (z,a,b,c,d,e,f,transient cycles to ignore(if plotting))
%   'freq'      (z,a,b,c,d,e,f,fft size,window size,resampling rate([] if none))
%   'lyapunov'  (parameter,parameter range [start step],parameter step,
%                   remaining variables structure)
%   'bifur'     (parameter,parameter range [start step],parameter step,
%                   remaining variables structure,transient cycles to ignore)
%   'phasemov'  (parameter,parameter range [start step],parameter step,
%                   remaining variables structure)
%   'freqmov'   (parameter,parameter range [start step],parameter step,
%                   remaining variables structure)
%   'poinmov'   (parameter,parameter range [start step],parameter step,
%                   remaining variables structure,transient cycles to ignore)
%
% Data is plotted if no output arguments are supplied for 'poincare' and
%   'freq'
% Ignored cycles for 'poincare' is only valid if data is plotted.
%-------------------------------------------------------------------------
function varargout = sync(flag,dir,varargin)
    global datadir
    tic;
    datadir = dir;

    if isa(flag,'char')~=1,
        error('No function flag specified!');
```

```
47      end
48      if isa(dir,'char')~=1,
49          error('No data directory specified!');
50      end
51
52      if strcmp(flag,'sim'),
53          if nargin~=13,
54              error('Invalid input arguments.  See help for details');
55          else
56              % check argument types
57              for i=3:nargin,
58                  if isa(varargin{i-2},'numeric'),
59                  else
60                      error('Invalid input arguments. Check datatype');
61                  end
62              end
63              timeSpan = varargin{1};
64              timeStep = varargin{2};
65              ICs      = varargin{3};
66              t_ignore = varargin{4};
67              z        = varargin{5};
68              a        = varargin{6};
69              b        = varargin{7};
70              c        = varargin{8};
71              d        = varargin{9};
72              e        = varargin{10};
73              f        = varargin{11};
74              options = struct('z',z,'a',a,'b',b,'c',c,'d',d,'e',e,'f',f,
75  ↪ 'timeSpan',timeSpan,'timeStep',timeStep,'ICs',ICs);
76
77              headwrite(1,flag,options);
78
79              X0 = [ICs(1) ICs(2) ICs(3) ICs(4) timeSpan(1)];
80              simflag(timeSpan,timeStep,X0,t_ignore,z,a,b,c,d,e,f);
81          end
82      end
83
84      if strcmp(flag,'poincare') | strcmp(flag,'freq'),
85          if nargin<10,
86              error('More input arguments required.  See help for details');
87          else
88              % check argument types
89              for i=3:nargin,
90                  if isa(varargin{i-2},'numeric'),
91                  else
92                      error('Invalid input arguments. Check datatype');
93                  end
```

```
94          end
95          z = varargin{1};
96          a = varargin{2};
97          b = varargin{3};
98          c = varargin{4};
99          d = varargin{5};
100         e = varargin{6};
101         f = varargin{7};
102         options = struct('z',z,'a',a,'b',b,'c',c,'d',d,'e',e,'f',f);
103         headwrite(1,flag,options);
104
105         if strcmp(flag,'poincare'),
106             c_ignore  = varargin{8};
107             [tp,x1p,x2p,x3p,x4p] = poincareflag(z,a,b,c,d,e,f);
108             if nargout==5,
109                 varargout{1} = tp;
110                 varargout{2} = x1p;
111                 varargout{3} = x2p;
112                 varargout{4} = x3p;
113                 varargout{5} = x4p;
114             end
115             if nargout==0;
116                 pmplotter(c_ignore,tp,x1p,x2p,x3p,x4p,options);
117             end
118         end
119
120         if strcmp(flag,'freq'),
121             if nargin<12,
122                 error('More input arguments required.  See help for details');
123             else
124                 nfft     = varargin{8};
125                 nwin     = varargin{9};
126                 sample   = varargin{10};
127                 if nfft < nwin,
128                     error('FFT length must be larger than window length!');
129                 end
130                 [t,x1,x3] = simloader(z,a,b,c,d,e,f);
131                 [Pxx,wx,Pyy,wy] = freqflag(t,x1,x3,nfft,nwin,sample);
132             end
133             if nargout==4,
134                 varargout{1} = Pxx;
135                 varargout{2} = wx;
136                 varargout{3} = Pyy;
137                 varargout{4} = wy;
138             end
139             if nargout==0;
140                 psdplotter(Pxx,wx,Pyy,wy,options);
```

```
141                 end
142             end
143         end
144     end
145
146     if strcmp(flag,'bifur') | strcmp(flag,'poinmov'),
147         if nargin<7,
148             error('More input arguments required.  See help for details');
149         else
150             var      = varargin{1};
151             varRange = varargin{2};
152             varStep  = varargin{3};
153             remain   = varargin{4};
154             c_ignore = varargin{5};
155             options = struct('var',var,'varRange',varRange,'varStep',varStep);
156             headwrite(1,flag,options);
157             if strcmp(flag,'bifur'),
158                 bifurflag(var,varRange,varStep,remain,c_ignore);
159             else
160                 poinmovflag(var,varRange,varStep,remain,c_ignore);
161             end
162         end
163     end
164
165     if strcmp(flag,'lyapunov') | strcmp(flag,'phasemov') | strcmp(flag,'freqmov'),
166         if nargin<6,
167             error('More input arguments required.  See help for details');
168         else
169             var      = varargin{1};
170             varRange = varargin{2};
171             varStep  = varargin{3};
172             remain   = varargin{4};
173             options = struct('var',var,'varRange',varRange,'varStep',varStep);
174             headwrite(1,flag,options);
175             if strcmp(flag,'lyapunov');
176                 lyapflag(var,varRange,varStep,remain);
177             elseif strcmp(flag,'phasemov');
178                 phasemovflag(var,varRange,varStep,remain);
179             else
180                 freqmovflag(var,varRange,varStep,remain);
181             end
182         end
183     end
184
185     headwrite(3,flag,[]);
186
187 return;
```

```
188
189 %-------------------------------------------------------------------------
190 % sim flag function
191 %-------------------------------------------------------------------------
192 function simflag(timeSpan,timeStep,X0,t_ignore,z,a,b,c,d,e,f);
193     global datadir
194     simfile = strcat(datadir,'\sim_sync_z',num2str(z),'_a',num2str(a),'_b',
195 ↪ num2str(b),'_c',num2str(c),'_d',num2str(d),'_e',num2str(e),'_f',
196 ↪ num2str(f),'.mat');
197     expfile = strcat(datadir,'\exp_sync_z',num2str(z),'_a',num2str(a),'_b',
198 ↪ num2str(b),'_c',num2str(c),'_d',num2str(d),'_e',num2str(e),'_f',
199 ↪ num2str(f),'.mat');
200     extern = cputime;
201     syncmex(timeSpan(2),timeStep,X0,t_ignore,z,a,b,c,d,e,f,simfile,expfile);
202     extern = cputime-extern;
203     periods = round((timeSpan(2)-timeSpan(1))/(2*pi));
204     transient = round(t_ignore/(2*pi));
205     options = struct('periods',periods,'externtime',extern,'transient',transient);
206     headwrite(2,'sim',options);
207 return;
208
209 %-------------------------------------------------------------------------
210 % poincare flag function
211 %-------------------------------------------------------------------------
212 function [tp,x1p,x2p,x3p,x4p] = poincareflag(z,a,b,c,d,e,f)
213     global datadir
214     simfile = strcat(datadir,'\sim_sync_z',num2str(z),'_a',num2str(a),'_b',
215 ↪ num2str(b),'_c',num2str(c),'_d',num2str(d),'_e',num2str(e),'_f',
216 ↪ num2str(f),'.mat');
217     poinfile = strcat(datadir,'\poincare_sync_z',num2str(z),'_a',num2str(a),
218 ↪ '_b',num2str(b),'_c',num2str(c),'_d',num2str(d),'_e',num2str(e),
219 ↪ '_f',num2str(f),'.mat');
220     extern = cputime;
221     [tp,x1p,x2p,x3p,x4p,periods] = poincare4mex(simfile,poinfile);
222     extern = cputime-extern;
223     options = struct('periods',periods,'externtime',extern);
224     headwrite(2,'poincare',options);
225 return;
226
227 %-------------------------------------------------------------------------
228 % freq flag function
229 %-------------------------------------------------------------------------
230 function [Pxx,wx,Pyy,wy] = freqflag(z,a,b,c,d,e,f)%(t,x1,x3,nfft,nwin,sample)
231     overlap = nwin/4;
232     datasize = 8*nwin-7*overlap;
233     ts  = t((end-datasize):end)-t(end-datasize);
234     x1s = x1((end-datasize):end);
```

```
235     x3s = x3((end-datasize):end);
236
237     % Resample if necessary
238     if isempty(sample),
239         sample = round(2*pi/(t(2)-t(1)));
240     else
241         in.time = ts;
242         in.signals.values = [x1s,x3s];
243         tStart = ts(1);
244         tStop = ts(datasize+1);
245         resampoptions = simset('FixedStep',2*pi/sample);
246         open_system('resample');
247         set_param('resample/To Workspace','VariableName','out');
248         [len,wid] = size(in.signals.values);
249         set_param('resample/In1','PortWidth',num2str(wid));
250         sim('resample',[tStart tStop],resampoptions,in);
251         save_system('resample');
252         close_system('resample');
253         x1s = out.signals.values(:,1);
254         x3s = out.signals.values(:,2);
255     end
256
257     % modify window type here
258     window = hamming(nwin);
259
260     [Pxx,wx] = psd(x1s,nfft,sample,window,overlap,'mean');
261     [Pyy,wy] = psd(x3s,nfft,sample,window,overlap,'mean');
262
263     options = struct('datasize',datasize,'nfft',nfft,'nwin',nwin,'overlap',
264 ↪ overlap,'sample',sample,'Pxx',Pxx,'wx',wx,'Pyy',Pyy,'wy',wy);
265     headwrite(2,'freq',options);
266 return;
267
268 %-------------------------------------------------------------------------
269 % lyapunov flag function
270 %-------------------------------------------------------------------------
271 function lyapflag(var,varRange,varStep,remain)
272     str = fieldnames(remain);
273     iterations = round((varRange(2)-varRange(1))/varStep)+1;
274
275     for i=1:iterations,
276         param(i) = varRange(1)+(i-1)*varStep;
277         filenames{i} = paramswitch('exp',var,param(i),remain);
278     end
279     extern = cputime;
280     [spec1,spec2,spec3,spec4,spec5,minEr,minEa] = lyapspecmex(filenames);
281     extern = cputime-extern;
```

```
282
283     figure(7);
284     hold on;
285     title('Lyapunov Spectrum');
286     xlabel(var);
287     ylabel('exponents (bit/sec)');
288
289     plot(param,spec1,param,spec2,param,spec3,param,spec4,param,spec5);
290     legend('\lambda_1','\lambda_2','\lambda_3','\lambda_4','\lambda_5',0);
291
292     v = axis;
293     if varStep<0,
294         axis([param(end) param(1) v(3) v(4)]);
295         xloc = (param(1)-param(end))*0.01+param(1);
296     else
297         axis([param(1) param(end) v(3) v(4)]);
298         xloc = (param(end)-param(1))*0.01+param(end);
299     end
300     yloc = (v(4)-v(3))/2+v(3);
301     yspace = (v(4)-v(3))*.05;
302
303     for i=1:length(str),
304         string = strcat(str{i},'=',num2str(remain.(str{i})));
305         text(xloc,(yloc-(i-1)*yspace),string);
306     end
307     options = struct('externtime',extern,'remain',remain,'minEr',minEr,
308 ↪ 'minEa',minEa);
309     headwrite(2,'lyapunov',options);
310
311 return;
312
313 %-------------------------------------------------------------------------
314 % bifurcation flag function
315 %-------------------------------------------------------------------------
316 function bifurflag(var,varRange,varStep,remain,c_ignore)
317     str = fieldnames(remain);
318     iterations = round((varRange(2)-varRange(1))/varStep)+1;
319
320     for i=1:iterations,
321         param(i) = varRange(1)+(i-1)*varStep;
322         filenames{i} = paramswitch('poincare',var,param(i),remain);
323     end
324     extern = cputime;
325     periods = bifurmex(param,c_ignore,filenames);
326     extern = cputime-extern;
327
328     figure(3);
```

```
329     title('Bifurcation Diagram (x1)');
330     xlabel(var);
331     ylabel('x1');
332     v = axis;
333     if varStep<0,
334         axis([param(end) param(1) v(3) v(4)]);
335         xloc = (param(1)-param(end))*0.01+param(1);
336     else
337         axis([param(1) param(end) v(3) v(4)]);
338         xloc = (param(end)-param(1))*0.01+param(end);
339     end
340     yloc = (v(4)-v(3))/2+v(3);
341     yspace = (v(4)-v(3))*.05;
342     for i=1:length(str),
343         string = strcat(str{i},'=',num2str(remain.(str{i})));
344         text(xloc,(yloc-(i-1)*yspace),string);
345     end
346
347     figure(4);
348     title('Bifurcation Diagram (x3)');
349     xlabel(var);
350     ylabel('x3');
351     v = axis;
352     if varStep<0,
353         axis([param(end) param(1) v(3) v(4)]);
354     else
355         axis([param(1) param(end) v(3) v(4)]);
356     end
357     yloc = (v(4)-v(3))/2+v(3);
358     yspace = (v(4)-v(3))*.05;
359     for i=1:length(str),
360         string = strcat(str{i},'=',num2str(remain.(str{i})));
361         text(xloc,(yloc-(i-1)*yspace),string);
362     end
363     options = struct('externtime',extern,'periods',periods,'transient',
364 ↪ c_ignore,'remain',remain);
365     headwrite(2,'bifur',options);
366 return;
367
368 %-------------------------------------------------------------------------
369 % phasemov flag function
370 %-------------------------------------------------------------------------
371 function phasemovflag(var,varRange,varStep,remain)
372     global datadir
373     nframes = round((varRange(2)-varRange(1))/varStep)+1;
374
375     for i=1:nframes,
```

```
376         param(i) = varRange(1)+(i-1)*varStep;
377         filenames{i} = paramswitch('sim',var,param(i),remain);
378         phasenames{i} = paramswitch('phase',var,param(i),remain);
379     end
380     extern = cputime;
381     [xplotsize,yplotsize] = phasemovmex(filenames,phasenames);
382     extern = cputime-extern;
383
384     str = strcat(var,'=',num2str(param(1)));
385     load(phasenames{1},'x1','x2','x3','x4');
386
387     gf = figure(12);
388     gp = plot(x1,x2,'b');
389     axis(xplotsize);
390     xhorzloc = xplotsize(1)+0.02*(xplotsize(2)-xplotsize(1));
391     xvertloc = xplotsize(4)-0.04*(xplotsize(4)-xplotsize(3));
392     gs = text(xhorzloc,xvertloc,str);
393     grect = get(gf,'position');
394     grect(1:2) = [0 0];
395     gname = strcat('PPX_',var,'_',num2str(varRange(1)),'-',num2str(varRange(2)),
396 ↪ '_',num2str(varStep),'.avi');
397     gavi = avifile(gname,'quality',100);
398
399     hf = figure(13);
400     hp = plot(x3,x4,'r');
401     axis(yplotsize);
402     yhorzloc = yplotsize(1)+0.02*(yplotsize(2)-yplotsize(1));
403     yvertloc = yplotsize(4)-0.04*(yplotsize(4)-yplotsize(3));
404     hs = text(yhorzloc,yvertloc,str);
405     hrect = get(hf,'position');
406     hrect(1:2) = [0 0];
407     hname = strcat('PPY_',var,'_',num2str(varRange(1)),'-',num2str(varRange(2)),
408 ↪ '_',num2str(varStep),'.avi');
409     havi = avifile(hname,'quality',100);
410
411     for i=1:nframes,
412         load(phasenames{i},'x1','x2','x3','x4');
413         str = strcat(var,'=',num2str(param(i)));
414         % Create new plots and strings, capture frames
415         set(gp,'XData',x1,'YData',x2);
416         set(gs,'string',str);
417         gavi = addframe(gavi,getframe(gf,grect));
418         set(hp,'XData',x3,'YData',x4);
419         set(hs,'string',str);
420         havi = addframe(havi,getframe(hf,hrect));
421     end
422     gavi = close(gavi);
```

```
423     havi = close(havi);
424     close(12);
425     close(13);
426     options = struct('externtime',extern,'remain',remain,'xmovname',gname,
427 ↪ 'ymovname',hname);
428     headwrite(2,'phasemov',options);
429 return;
430
431 %-------------------------------------------------------------------------
432 % poinmov flag function
433 %-------------------------------------------------------------------------
434 function poinmovflag(var,varRange,varStep,remain,c_ignore)
435     global datadir
436     nframes = round((varRange(2)-varRange(1))/varStep)+1;
437
438     for i=1:nframes,
439         param(i) = varRange(1)+(i-1)*varStep;
440         filenames{i} = paramswitch('poincare',var,param(i),remain);
441         tempnames{i} = strcat(datadir,'\temp',num2str(i),'.mat');
442     end
443     extern = cputime;
444     [xplotsize,yplotsize,periods] = poinmovmex(c_ignore,filenames,tempnames);
445     extern = cputime-extern;
446
447     str = strcat(var,'=',num2str(param(1)));
448     load(tempnames{1},'x1p','x2p','x3p','x4p');
449
450     gf = figure(8);
451     gp = plot(x1p,x2p,'b.','markersize',5);
452     axis(xplotsize);
453     xhorzloc = xplotsize(1)+0.02*(xplotsize(2)-xplotsize(1));
454     xvertloc = xplotsize(4)-0.04*(xplotsize(4)-xplotsize(3));
455     gs = text(xhorzloc,xvertloc,str);
456     grect = get(gf,'position');
457     grect(1:2) = [0 0];
458     gname = strcat('PMX_',var,'_',num2str(varRange(1)),'-',num2str(varRange(2)),
459 ↪ '_',num2str(varStep),'.avi');
460     gavi = avifile(gname,'quality',100);
461
462     hf = figure(9);
463     hp = plot(x3p,x4p,'r.','markersize',5);
464     axis(yplotsize);
465     yhorzloc = yplotsize(1)+0.02*(yplotsize(2)-yplotsize(1));
466     yvertloc = yplotsize(4)-0.04*(yplotsize(4)-yplotsize(3));
467     hs = text(yhorzloc,yvertloc,str);
468     hrect = get(hf,'position');
469     hrect(1:2) = [0 0];
```

```
    hname = strcat('PMY_',var,'_',num2str(varRange(1)),'-',num2str(varRange(2)),
↪ '_',num2str(varStep),'.avi');
    havi = avifile(hname,'quality',100);

    for i=1:nframes,
        load(tempnames{i},'x1p','x2p','x3p','x4p');
        delete(tempnames{i});
        str = strcat(var,'=',num2str(param(i)));
        % Create new plots and strings, capture frames
        set(gp,'XData',x1p,'YData',x2p);
        set(gs,'string',str);
        gavi = addframe(gavi,getframe(gf,grect));
        set(hp,'XData',x3p,'YData',x4p);
        set(hs,'string',str);
        havi = addframe(havi,getframe(hf,hrect));
    end
    gavi = close(gavi);
    havi = close(havi);
    close(8);
    close(9);
    options = struct('externtime',extern,'periods',periods,'transient',c_ignore,
↪ 'remain',remain,'xmovname',gname,'ymovname',hname);
    headwrite(2,'poinmov',options);
return;

%-------------------------------------------------------------------------
% freqmov flag function
%-------------------------------------------------------------------------
function freqmovflag(var,varRange,varStep,remain)
    nframes = round((varRange(2)-varRange(1))/varStep)+1;
    param(1) = varRange(1);
    str = strcat(var,'=',num2str(param(1)));
    [wx,Pxx,wy,Pyy] = paramswitch('freq',var,param(1),remain);

    gf = figure(10);
    gp = plot(wx,10*log10(Pxx),'b');
    axis([0 1 -50 40]);
    horzloc = .025;
    vertloc = 35;
    gs = text(horzloc,vertloc,str);
    grect = get(gf,'position');
    grect(1:2) = [0 0];
    gname = strcat('FMX_',var,'_',num2str(varRange(1)),'-',num2str(varRange(2)),
↪ '_',num2str(varStep),'.avi');
    gavi = avifile(gname,'quality',100);

    hf = figure(11);
```

```
517     hp = plot(wy,10*log10(Pyy),'r');
518     axis([0 1 -50 40]);
519     hs = text(horzloc,vertloc,str);
520     hrect = get(hf,'position');
521     hrect(1:2) = [0 0];
522     hname = strcat('FMY_',var,'_',num2str(varRange(1)),'-',num2str(varRange(2)),
523 ↪ '_',num2str(varStep),'.avi');
524     havi = avifile(hname,'quality',100);
525
526     for i=1:nframes,
527         param(i) = varRange(1)+(i-1)*varStep;
528         [wx,Pxx,wy,Pyy] = paramswitch('freq',var,param(i),remain);
529         str = strcat(var,'=',num2str(param(i)));
530         % Create new plots and strings, capture frames
531         set(gp,'XData',wx,'YData',10*log10(Pxx));
532         set(gs,'string',str);
533         gavi = addframe(gavi,getframe(gf,grect));
534         set(hp,'XData',wy,'YData',10*log10(Pyy));
535         set(hs,'string',str);
536         havi = addframe(havi,getframe(hf,hrect));
537     end
538     gavi = close(gavi);
539     havi = close(havi);
540     close(10);
541     close(11);
542     options = struct('remain',remain,'xmovname',gname,'ymovname',hname);
543     headwrite(2,'freqmov',options);
544 return;
545
546 %-------------------------------------------------------------------------
547 % Header file writer
548 %-------------------------------------------------------------------------
549 function headwrite(pass,flag,options)
550     global datadir
551     persistent fid outfile
552
553     workdir = pwd;
554     cd(datadir);
555
556     switch pass
557         case 1
558             % Open and write initial header
559             if strcmp(flag,'bifur') | strcmp(flag,'lyapunov') |
560 ↪ strcmp(flag,'phasemov') | strcmp(flag,'poinmov') | strcmp(flag,'freqmov'),
561                 outfile = strcat(flag,'_sync_',options.var,'_',
562 ↪ num2str(options.varRange(1)),'-',num2str(options.varRange(2)),'_',
563 ↪ num2str(options.varStep));
```

```
564             else
565                 outfile = strcat(flag,'_sync_z',num2str(options.z),'_a',
566 ↪ num2str(options.a),'_b',num2str(options.b),'_c',num2str(options.c),
567 ↪ '_d',num2str(options.d),'_e',num2str(options.e),'_f',num2str(options.f));
568             end
569             fid = fopen(strcat(outfile,'.txt'),'wt');
570             fprintf(fid,'-------------------------------------------------
571 ↪ ------------------------');
572             fprintf(fid,'\n Simulation of nonlinearly coupled Duffing Oscillator
573 ↪ system with');
574             fprintf(fid,'\n    synchronous forcing');
575             fprintf(fid,'\n by Joseph O''Day, 2005');
576             fprintf(fid,'\n-------------------------------------------------
577 ↪ -------------------------');
578             today = date;
579             now = clock;
580             fprintf(fid,'\n Run Date: %s',today);
581             fprintf(fid,'\n Run Time: %i:%i',now(4),now(5));
582             fprintf(fid,'\n\n Output results from flag type: %s',flag);
583
584             if strcmp(flag,'bifur') | strcmp(flag,'lyapunov') |
585 ↪ strcmp(flag,'phasemov') | strcmp(flag,'poinmov') | strcmp(flag,'freqmov'),
586                 if strcmp(flag,'bifur') | strcmp(flag,'lyapunov'),
587                     fprintf(fid,'\n\n Diagram Parameters:');
588                 else
589                     fprintf(fid,'\n\n Movie Parameters:');
590                 end
591                 fprintf(fid,'\n\t Variable:       %s',options.var);
592                 fprintf(fid,'\n\t Variable Range: %.3f-%.3f',options.varRange(1),
593 ↪ options.varRange(2));
594                 fprintf(fid,'\n\t Variable Step:  %.3f',options.varStep);
595             else
596                 % Print system parameters, ICs, time span
597                 fprintf(fid,'\n\n System Parameters:');
598                 fprintf(fid,'\n\t z = %.3f',options.z);
599                 fprintf(fid,'\n\t a = %.3f',options.a);
600                 fprintf(fid,'\n\t b = %.3f',options.b);
601                 fprintf(fid,'\n\t c = %.3f',options.c);
602                 fprintf(fid,'\n\t d = %.3f',options.d);
603                 fprintf(fid,'\n\t e = %.3f',options.e);
604                 fprintf(fid,'\n\t f = %.3f',options.f);
605                 if strcmp(flag,'sim'),
606                     fprintf(fid,'\n\n Initial Conditions:');
607                     fprintf(fid,'\n\t x1 = %.3f',options.ICs(1));
608                     fprintf(fid,'\n\t x2 = %.3f',options.ICs(2));
609                     fprintf(fid,'\n\t x3 = %.3f',options.ICs(3));
610                     fprintf(fid,'\n\t x4 = %.3f',options.ICs(4));
```

```
611                     fprintf(fid,'\n\n Simulation Parameters:');
612                     fprintf(fid,'\n\t Time Span:         %.3f-%.3f seconds',
613 ↪ options.timeSpan(1),options.timeSpan(2));
614                     fprintf(fid,'\n\t Step Size:         %.3f seconds',
615 ↪ options.timeStep);
616                 end
617             end
618
619         case 2
620             % Print flag type data & save
621             if strcmp(flag,'sim'),
622                 fprintf(fid,'\n\t Transient Periods: %d',options.transient);
623                 fprintf(fid,'\n\t Total Periods:     %d',options.periods);
624                 time = toc;
625                 fprintf(fid,'\n\n Total execution time required:  %d minutes
626 ↪ %.2f seconds',floor(time/60),time-floor(time/60)*60);
627                 extern = options.externtime;
628                 fprintf(fid,'\n Total time spent in SYNCMEX.C:  %d minutes
629 ↪ %.2f seconds',floor(extern/60),extern-floor(extern/60)*60);
630                 [token,rem] = strtok(outfile,'_');
631                 datfile = rem;
632                 fprintf(fid,'\n\n Time series output file saved as:
633 ↪ %s',strcat('sim',datfile,'.mat'));
634                 fprintf(fid,'\n Lyapunov exponents output file saved as: %s',
635 ↪ strcat('exp',datfile,'.mat'));
636             end
637             if strcmp(flag,'poincare'),
638                 [token,rem] = strtok(outfile,'_');
639                 datfile = rem;
640                 fprintf(fid,'\n\n Mapped data from: %s',strcat('sim',datfile,
641 ↪ '.mat'));
642                 fprintf(fid,'\n\n Total Periods: %d',options.periods);
643                 time = toc;
644                 fprintf(fid,'\n\n Total mapping time required:         %d minutes
645 ↪ %.2f seconds',floor(time/60),time-floor(time/60)*60);
646                 extern = options.externtime;
647                 fprintf(fid,'\n Total time spent in POINCARE4MEX.C:  %d minutes
648 ↪ %.2f seconds',floor(extern/60),extern-floor(extern/60)*60);
649             end
650             if strcmp(flag,'freq'),
651                 [token,rem] = strtok(outfile,'_');
652                 datfile = rem;
653                 fprintf(fid,'\n PSD performed on time series data from: %s',
654 ↪ strcat('sim',datfile,'.mat'));
655                 fprintf(fid,'\n\n FFT Parameters:');
656                 fprintf(fid,'\n\t Datasize:      %d',options.datasize);
657                 fprintf(fid,'\n\t FFT size:      %d',options.nfft);
```

```
658                 fprintf(fid,'\n\t Window type:   Hamming');
659                 fprintf(fid,'\n\t Window size:   %d',options.nwin);
660                 fprintf(fid,'\n\t Overlaps:      %d',options.overlap);
661                 fprintf(fid,'\n\t Sampling rate: %.3f',options.sample);
662                 time = toc;
663                 fprintf(fid,'\n\n Total calculation time required:  %d minutes
664 ↪ %.2f seconds',floor(time/60),time-floor(time/60)*60);
665                 Pxx = options.Pxx;
666                 wx = options.wx;
667                 Pyy = options.Pyy;
668                 wy = options.wy;
669                 save(strcat(outfile,'.mat'),'Pxx','wx','Pyy','wy','outfile');
670             end
671             if strcmp(flag,'bifur') | strcmp(flag,'lyapunov') |
672 ↪ strcmp(flag,'phasemov') | strcmp(flag,'poinmov') | strcmp(flag,'freqmov'),
673                 fprintf(fid,'\n\n Remaining system parameters:');
674                 params = fieldnames(options.remain);
675                 n = length(params);
676                 for i=1:n,
677                     fprintf(fid,'\n\t %s = %.3f',params{i},
678 ↪ options.remain.(params{i}));
679                 end
680                 if strcmp(flag,'lyapunov'),
681                     time = toc;
682                     fprintf(fid,'\n\n Total spectrum creation time required:
683 ↪ %d minutes %.2f seconds',floor(time/60),time-floor(time/60)*60);
684                     extern = options.externtime;
685                     fprintf(fid,'\n Total time spent in LYAPSPECMEX.C:
686 ↪ %d minutes %.2f seconds',floor(extern/60),extern-floor(extern/60)*60);
687                     fprintf(fid,'\n\n Minimum Relative Convergence:  %.4e',
688 ↪ options.minEr);
689                     fprintf(fid,'\n Minimum Absolute Convergence:  %.4e',
690 ↪ options.minEa);
691                     x=figure(7);
692                     xfig = strcat(outfile,'.fig');
693                     saveas(x,xfig);
694                     fprintf(fid,'\n\n\n Output figure file saved as: %s',xfig);
695                 end
696                 if strcmp(flag,'bifur') | strcmp(flag,'poinmov'),
697                     fprintf(fid,'\n\n Based on Poincare maps with:');
698                     fprintf(fid,'\n\t Total Periods:      %d',options.periods);
699                     fprintf(fid,'\n\t Transient Periods:  %d',options.transient);
700                     if strcmp(flag,'bifur'),
701                         time = toc;
702                         fprintf(fid,'\n\n Total diagram creation time required:
703 ↪ %d minutes %.2f seconds',floor(time/60),time-floor(time/60)*60);
704                         extern = options.externtime;
```

```
                        fprintf(fid,'\n Total time spent in BIFURMEX.C:
↪ %d minutes %.2f seconds',floor(extern/60),extern-floor(extern/60)*60);
                        x=figure(3);
                        y=figure(4);
                        xfig = strcat(outfile,'_x.fig');
                        yfig = strcat(outfile,'_y.fig');
                        saveas(x,xfig);
                        saveas(y,yfig);
                        fprintf(fid,'\n\n\n Output figure files saved as: %s',
↪ xfig);
                        fprintf(fid,'\n                                    %s',yfig);
                    end
                    if strcmp(flag,'poinmov'),
                        time = toc;
                        fprintf(fid,'\n\n Total movie creation time required:
↪ %d minutes %.2f seconds',floor(time/60),time-floor(time/60)*60);
                        extern = options.externtime;
                        fprintf(fid,'\n Total time spent in POINMOVMEX.C:
↪ %d minutes %.2f seconds',floor(extern/60),extern-floor(extern/60)*60);
                    end
                end
                if strcmp(flag,'phasemov'),
                    time = toc;
                    fprintf(fid,'\n\n Total movie creation time required:
↪ %d minutes %.2f seconds',floor(time/60),time-floor(time/60)*60);
                    extern = options.externtime;
                    fprintf(fid,'\n Total time spent in PHASEMOVMEX.C:   %d
↪ minutes %.2f seconds',floor(extern/60),extern-floor(extern/60)*60);
                end
                if strcmp(flag,'freqmov'),
                    time = toc;
                    fprintf(fid,'\n\n Total movie creation time required:
↪ %d minutes %.2f seconds',floor(time/60),time-floor(time/60)*60);
                end
            end
            if strcmp(flag,'poincare') | strcmp(flag,'freq'),
                fprintf(fid,'\n\n\n Output file saved as: %s',
↪ strcat(outfile,'.mat'));
            end
            if strcmp(flag,'phasemov') | strcmp(flag,'poinmov') |
↪ strcmp(flag,'freqmov'),
                fprintf(fid,'\n\n Output .avi files saved as: %s',
↪ options.xmovname);
                fprintf(fid,'\n                                    %s',
↪ options.ymovname);
            end

```

```
752         case 3
753             % Termination
754             now = clock;
755             fprintf(fid,'\n\n\n Normal Termination of SYNC.M');
756             fprintf(fid,'\n End Time: %i:%i',now(4),now(5));
757             fprintf(fid,'\n-------------------------------------------
758 ↪ -------------------------');
759             fclose(fid);
760 
761         otherwise
762             error('Too many function calls to HEADWRITE');
763     end
764     cd(workdir);
765 return;
766 
767 %-------------------------------------------------------------------------
768 % Poincare Map plotter function
769 %-------------------------------------------------------------------------
770 function pmplotter(c_ignore,tp,x1p,x2p,x3p,x4p,options);
771     cycles = length(x1p)-1;
772     x1p = x1p(c_ignore+1:end);
773     x2p = x2p(c_ignore+1:end);
774     x3p = x3p(c_ignore+1:end);
775     x4p = x4p(c_ignore+1:end);
776 
777     str = fieldnames(options);
778 
779     figure(1);
780     plot(x1p,x2p,'b.','markersize',2);
781     title(['Poincare Map (',num2str(cycles),' forcing cycles, ',num2str(c_ignore),
782 ↪ ' cycles ignored as transients)']);
783     xlabel('x1');
784     ylabel('x2');
785     v = axis;
786     xloc = (v(2)-v(1))*0.01+v(2);
787     yloc = (v(4)-v(3))/2+v(3);
788     yspace = (v(4)-v(3))*0.05;
789     for i=1:length(str),
790         string = strcat(str{i},'=',num2str(options.(str{i})));
791         text(xloc,(yloc-(i-1)*yspace),string);
792     end
793 
794     figure(2);
795     plot(x3p,x4p,'r.','markersize',2);
796     title(['Poincare Map (',num2str(cycles),' forcing cycles, ',num2str(c_ignore),
797 ↪ ' cycles ignored as transients)']);
798     xlabel('x3');
```

```
799       ylabel('x4');
800       v = axis;
801       xloc = (v(2)-v(1))*0.01+v(2);
802       yloc = (v(4)-v(3))/2+v(3);
803       yspace = (v(4)-v(3))*0.05;
804       for i=1:length(str),
805           string = strcat(str{i},'=',num2str(options.(str{i})));
806           text(xloc,(yloc-(i-1)*yspace),string);
807       end
808
809   return;
810
811   %-------------------------------------------------------------------------
812   % PSD plotter function
813   %-------------------------------------------------------------------------
814   function psdplotter(Pxx,wx,Pyy,wy,options)
815       str = fieldnames(options);
816       xloc = 1.01;
817       yloc = -5;
818       yspace = 4.5;
819
820       figure(5);
821       plot(wx,10*log10(Pxx),'b');
822       xlabel('Angular Frequency (rad/s)');
823       ylabel('Amplitude (dB)');
824       title('PSD (x1-direction)');
825       axis([0 1 -50 40]);
826       for i=1:length(str),
827           string = strcat(str{i},'=',num2str(options.(str{i})));
828           text(xloc,(yloc-(i-1)*yspace),string);
829       end
830
831       figure(6);
832       plot(wy,10*log10(Pyy),'r');
833       xlabel('Angular Frequency (rad/s)');
834       ylabel('Amplitude (dB)');
835       title('PSD (x3-direction)');
836       axis([0 1 -50 40]);
837       for i=1:length(str),
838           string = strcat(str{i},'=',num2str(options.(str{i})));
839           text(xloc,(yloc-(i-1)*yspace),string);
840       end
841
842   return;
843
844   %-------------------------------------------------------------------------
845   % Parameter switching for file naming
```

```
%-------------------------------------------------------------------------
function varargout = paramswitch(flag,var,value,remain)
    global datadir
    if abs(value)<1e-9,
        value=0;
    end
    switch var
        case 'a'
            filename = strcat(datadir,'\',flag,'_sync_z',num2str(remain.z),
↪ '_a',num2str(value),'_b',num2str(remain.b),'_c',num2str(remain.c),
↪ '_d',num2str(remain.d),'_e',num2str(remain.e),'_f',num2str(remain.f),'.mat');
        case 'b'
            filename = strcat(datadir,'\',flag,'_sync_z',num2str(remain.z),
↪ '_a',num2str(remain.a),'_b',num2str(value),'_c',num2str(remain.c),
↪ '_d',num2str(remain.d),'_e',num2str(remain.e),'_f',num2str(remain.f),'.mat');
        case 'c'
            filename = strcat(datadir,'\',flag,'_sync_z',num2str(remain.z),
↪ '_a',num2str(remain.a),'_b',num2str(remain.b),'_c',num2str(value),
↪ '_d',num2str(remain.d),'_e',num2str(remain.e),'_f',num2str(remain.f),'.mat');
        case 'd'
            filename = strcat(datadir,'\',flag,'_sync_z',num2str(remain.z),
↪ '_a',num2str(remain.a),'_b',num2str(remain.b),'_c',num2str(remain.c),
↪ '_d',num2str(value),'_e',num2str(remain.e),'_f',num2str(remain.f),'.mat');
        case 'e'
            filename = strcat(datadir,'\',flag,'_sync_z',num2str(remain.z),
↪ '_a',num2str(remain.a),'_b',num2str(remain.b),'_c',num2str(remain.c),
↪ '_d',num2str(remain.d),'_e',num2str(value),'_f',num2str(remain.f),'.mat');
        case 'f'
            filename = strcat(datadir,'\',flag,'_sync_z',num2str(remain.z),
↪ '_a',num2str(remain.a),'_b',num2str(remain.b),'_c',num2str(remain.c),
↪ '_d',num2str(remain.d),'_e',num2str(remain.e),'_f',num2str(value),'.mat');
        case 'a-c'
            filename = strcat(datadir,'\',flag,'_sync_z',num2str(remain.z),
↪ '_a',num2str(value),'_b',num2str(remain.b),'_c',num2str(value),
↪ '_d',num2str(remain.d),'_e',num2str(remain.e),'_f',num2str(remain.f),'.mat');
        case 'b-d'
            filename = strcat(datadir,'\',flag,'_sync_z',num2str(remain.z),
↪ '_a',num2str(remain.a),'_b',num2str(value),'_c',num2str(remain.c),
↪ '_d',num2str(value),'_e',num2str(remain.e),'_f',num2str(remain.f),'.mat');
        otherwise
            error('Invalid parameter');
    end
    if strcmp(flag,'freq'),
        load(filename,'wx','Pxx','wy','Pyy');
        nargout = 4;
        varargout{1} = wx;
        varargout{2} = Pxx;
```

```
893         varargout{3} = wy;
894         varargout{4} = Pyy;
895     else
896         nargout = 1;
897         varargout{1} = filename;
898     end
899
900 return;
901
902 %-------------------------------------------------------------------------
903 % Sim Data loader
904 %-------------------------------------------------------------------------
905 function [t,x1,x3] = simloader(z,a,b,c,d,e,f)
906     global datadir
907
908     workdir = pwd;
909     cd(datadir);
910     infile = strcat('sim_sync_z',num2str(z),'_a',num2str(a),'_b',num2str(b),
911 ↪ '_c',num2str(c),'_d',num2str(d),'_e',num2str(e),'_f',num2str(f),'.mat');
912     nargout = 3;
913     load(infile,'t','x1','x3');
914     varargout{1} = t;
915     varargout{2} = x1;
916     varargout{3} = x3;
917     cd(workdir);
918 return;
```

Appendix D

Program Outputs

D.1 Flag Output - 'sim'

```
1  ------------------------------------------------------------------------
2   Simulation of nonlinearly coupled Duffing Oscillator system with
3      synchronous forcing
4   by Joseph O'Day, 2005
5  ------------------------------------------------------------------------
6   Run Date: 19-Sep-2005
7   Run Time: 14:47
8
9   Output results from flag type: sim
10
11  System Parameters:
12      z = 0.300
13      a = -1.500
14      b = 0.100
15      c = -2.000
16      d = 0.500
17      e = 0.001
18      f = 1.500
19
20  Initial Conditions:
21      x1 = 4.741
22      x2 = 0.102
23      x3 = 1.574
24      x4 = 2.863
25
26  Simulation Parameters:
27      Time Span:          0.000-18849.556 seconds
28      Step Size:          0.013 seconds
29      Transient Periods: 500
30      Total Periods:      3000
31
```

```
Total execution time required:  0 minutes 21.56 seconds
Total time spent in SYNCMEX.C:  0 minutes 19.80 seconds

Time series output file saved as:
           sim_sync_z0.3_a-1.5_b0.1_c-2_d0.5_e0.001_f1.5.mat
Lyapunov exponents output file saved as:
           exp_sync_z0.3_a-1.5_b0.1_c-2_d0.5_e0.001_f1.5.mat


Normal Termination of SYNC.M
End Time: 14:48
-------------------------------------------------------------------------
```

D.2 Flag Output - 'poincare'

```
1  ------------------------------------------------------------------------
2   Simulation of nonlinearly coupled Duffing Oscillator system with
3      synchronous forcing
4   by Joseph O'Day, 2005
5  ------------------------------------------------------------------------
6   Run Date: 19-Sep-2005
7   Run Time: 14:49
8
9   Output results from flag type: poincare
10
11  System Parameters:
12      z = 0.300
13      a = -1.500
14      b = 0.100
15      c = -2.000
16      d = 0.500
17      e = 0.003
18      f = 1.500
19
20  Mapped data from: sim_sync_z0.3_a-1.5_b0.1_c-2_d0.5_e0.003_f1.5.mat
21
22  Total Periods: 3000
23
24  Total mapping time required:         0 minutes 0.23 seconds
25  Total time spent in POINCARE4MEX.C:  0 minutes 0.19 seconds
26
27  Output file saved as:
28             poincare_sync_z0.3_a-1.5_b0.1_c-2_d0.5_e0.003_f1.5.mat
29
30
31  Normal Termination of SYNC.M
32  End Time: 14:49
33 ------------------------------------------------------------------------
```

D.3 Flag Output - 'freq'

```
1  ------------------------------------------------------------------------
2   Simulation of nonlinearly coupled Duffing Oscillator system with
3      synchronous forcing
4   by Joseph O'Day, 2005
5  ------------------------------------------------------------------------
6   Run Date: 19-Sep-2005
7   Run Time: 14:48
8
9   Output results from flag type: freq
10
11  System Parameters:
12      z = 0.300
13      a = -1.500
14      b = 0.100
15      c = -2.000
16      d = 0.500
17      e = 0.001
18      f = 1.500
19  PSD performed on time series data from:
20              sim_sync_z0.3_a-1.5_b0.1_c-2_d0.5_e0.001_f1.5.mat
21
22  FFT Parameters:
23      Datasize:       409600
24      FFT size:       65536
25      Window type:    Hamming
26      Window size:    65536
27      Overlaps:       16384
28      Sampling rate: 500.000
29
30  Total calculation time required:  0 minutes 0.91 seconds
31
32  Output file saved as: freq_sync_z0.3_a-1.5_b0.1_c-2_d0.5_e0.001_f1.5.mat
33
34
35  Normal Termination of SYNC.M
36  End Time: 14:48
37 ------------------------------------------------------------------------
```

D.4 Flag Output - 'bifur'

```
1  ------------------------------------------------------------------------
2   Simulation of nonlinearly coupled Duffing Oscillator system with
3      synchronous forcing
4   by Joseph O'Day, 2005
5  ------------------------------------------------------------------------
6   Run Date: 20-Sep-2005
7   Run Time: 10:48
8
9   Output results from flag type: bifur
10
11  Diagram Parameters:
12      Variable:       e
13      Variable Range: 0.000-0.900
14      Variable Step:  0.001
15
16  Remaining system parameters:
17      z = 0.300
18      a = -1.500
19      b = 0.100
20      c = -2.000
21      d = 0.500
22      f = 1.500
23
24  Based on Poincare maps with:
25      Total Periods:      3000
26      Transient Periods:  500
27
28  Total diagram creation time required:  0 minutes 57.52 seconds
29  Total time spent in BIFURMEX.C:        0 minutes 1.31 seconds
30
31  Output figure files saved as: bifur_sync_e_0-0.9_0.001_x.fig
32                                bifur_sync_e_0-0.9_0.001_y.fig
33
34
35  Normal Termination of SYNC.M
36  End Time: 10:50
37 ------------------------------------------------------------------------
```

D.5 Flag Output - 'lyapunov'

```
1  ------------------------------------------------------------------------
2   Simulation of nonlinearly coupled Duffing Oscillator system with
3      synchronous forcing
4   by Joseph O'Day, 2005
5  ------------------------------------------------------------------------
6   Run Date: 19-Sep-2005
7   Run Time: 16:50
8
9   Output results from flag type: lyapunov
10
11  Diagram Parameters:
12      Variable:       e
13      Variable Range: 0.000-0.310
14      Variable Step:  0.001
15
16  Remaining system parameters:
17      z = 0.300
18      a = -1.500
19      b = 0.100
20      c = -2.000
21      d = 0.500
22      f = 1.500
23
24  Total spectrum creation time required:  0 minutes 25.64 seconds
25  Total time spent in LYAPSPECMEX.C:      0 minutes 0.55 seconds
26
27  Minimum Relative Convergence:  1.7686e-002
28  Minimum Absolute Convergence:  3.6874e-006
29
30  Output figure file saved as: lyapunov_sync_e_0-0.31_0.001.fig
31
32
33  Normal Termination of SYNC.M
34  End Time: 16:50
35 ------------------------------------------------------------------------
```

D.6 Flag Output - 'phasemov'

```
1  ------------------------------------------------------------------------
2   Simulation of nonlinearly coupled Duffing Oscillator system with
3      synchronous forcing
4   by Joseph O'Day, 2005
5  ------------------------------------------------------------------------
6   Run Date: 21-Sep-2005
7   Run Time: 1:36
8
9   Output results from flag type: phasemov
10
11  Movie Parameters:
12      Variable:       e
13      Variable Range: 0.000-0.900
14      Variable Step:  0.001
15
16  Remaining system parameters:
17      z = 0.300
18      a = -1.500
19      b = 0.100
20      c = -2.000
21      d = 0.500
22      f = 1.500
23
24  Total movie creation time required:  6 minutes 15.20 seconds
25  Total time spent in PHASEMOVMEX.C:   0 minutes 1.61 seconds
26
27  Output .avi files saved as: PPX_e_0-0.9_0.001.avi
28                              PPY_e_0-0.9_0.001.avi
29
30
31  Normal Termination of SYNC.M
32  End Time: 1:43
33 ------------------------------------------------------------------------
```

D.7 Flag Output - 'poinmov'

```
1  ------------------------------------------------------------------------
2   Simulation of nonlinearly coupled Duffing Oscillator system with
3      synchronous forcing
4   by Joseph O'Day, 2005
5  ------------------------------------------------------------------------
6   Run Date: 19-Sep-2005
7   Run Time: 17:1
8
9   Output results from flag type: poinmov
10
11  Movie Parameters:
12      Variable:       e
13      Variable Range: 0.000-0.310
14      Variable Step:  0.001
15
16  Remaining system parameters:
17      z = 0.300
18      a = -1.500
19      b = 0.100
20      c = -2.000
21      d = 0.500
22      f = 1.500
23
24  Based on Poincare maps with:
25      Total Periods:      3000
26      Transient Periods:  500
27
28  Total movie creation time required:  5 minutes 46.68 seconds
29  Total time spent in POINMOVMEX.C:    0 minutes 0.69 seconds
30
31  Output .avi files saved as: PMX_e_0-0.31_0.001.avi
32                              PMY_e_0-0.31_0.001.avi
33
34
35  Normal Termination of SYNC.M
36  End Time: 17:8
37 ------------------------------------------------------------------------
```

D.8 Flag Output - 'freqmov'

```
1  ------------------------------------------------------------------------
2   Simulation of nonlinearly coupled Duffing Oscillator system with
3      synchronous forcing
4   by Joseph O'Day, 2005
5  ------------------------------------------------------------------------
6   Run Date: 21-Sep-2005
7   Run Time: 1:51
8
9   Output results from flag type: freqmov
10
11  Movie Parameters:
12      Variable:       e
13      Variable Range: 0.000-0.900
14      Variable Step:  0.001
15
16  Remaining system parameters:
17      z = 0.300
18      a = -1.500
19      b = 0.100
20      c = -2.000
21      d = 0.500
22      f = 1.500
23
24  Total movie creation time required:  8 minutes 8.18 seconds
25
26  Output .avi files saved as: FMX_e_0-0.9_0.001.avi
27                              FMY_e_0-0.9_0.001.avi
28
29
30  Normal Termination of SYNC.M
31  End Time: 1:59
32 ------------------------------------------------------------------------
```